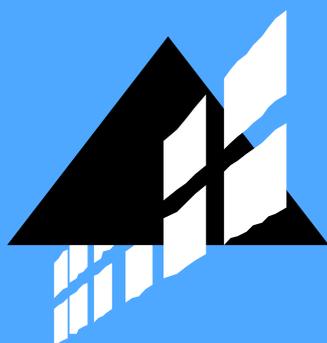
Manual de Utilização AL-2005/RTMP

Rev. E 05/2006

Cód. Doc.: MU207006

Conforme o Código de Defesa do Consumidor vigente no Brasil, informamos a seguir, aos clientes que utilizam nossos produtos, aspectos relacionados com a segurança de pessoas e instalações.

Os equipamentos de automação industrial fabricados pela Altus são robustos e confiáveis devido ao rígido controle de qualidade a que são submetidos. No entanto, equipamentos eletrônicos de controle industrial (controladores programáveis, comandos numéricos, etc.) podem causar danos às máquinas ou processos por eles controlados em caso de defeito em suas partes e peças ou de erros de programação ou instalação, podendo inclusive colocar em risco vidas humanas.

O usuário deve analisar as possíveis conseqüências destes defeitos e providenciar instalações adicionais externas de segurança que, em caso de necessidade, sirvam para preservar a segurança do sistema, principalmente nos casos da instalação inicial e de testes.

É imprescindível a leitura completa dos manuais e/ou características técnicas do produto antes da instalação ou utilização do mesmo.

A Altus garante os seus equipamentos conforme descrito nas Condições Gerais de Fornecimento, anexada às propostas comerciais.

A Altus garante que seus equipamentos funcionam de acordo com as descrições contidas explicitamente em seus manuais e/ou características técnicas, não garantindo a satisfação de algum tipo particular de aplicação dos equipamentos.

A Altus desconsiderará qualquer outra garantia, direta ou implícita, principalmente quando se tratar de fornecimento de terceiros.

Pedidos de informações adicionais sobre o fornecimento e/ou características dos equipamentos e serviços Altus devem ser feitos por escrito. A Altus não se responsabiliza por informações fornecidas sobre seus equipamentos sem registro formal.

DIREITOS AUTORAIS

Sumário

1. Introdução

O Real-Time Multitasking Processor

O Real-Time Multitasking Processor AL-2005 é um processador de comunicação e de algoritmos que opera em conjunto com as UCPs AL-2003 ou AL-2004.

O Processador Multitarefa AL-2005 pode executar tarefas diversas tais como comunicação com outros equipamentos utilizando protocolos quaisquer e programas aplicativos escritos em linguagem de alto nível (linguagem "C").

Os aplicativos do processador multitarefa são desenvolvidos utilizando-se uma biblioteca de funções própria do AL-2005.

O processador multitarefa AL-2005 possui sistema operacional multitarefa de tempo real preemptivo que permite a execução de vários módulos de programa simultaneamente, acessando a memória imagem da UCP de forma concorrente.

O AL-2005 permite que sejam acoplados até dois módulos seriais. Cada um destes módulos implementa um padrão físico de comunicação diferente (AL-2405/485I para o padrão RS-485, AL-2405/232 para o padrão RS-232C).

Sob o ponto de vista do programa aplicativo no CP, toda a comunicação entre o processador Multitarefa AL-2005 e a UCP é realizada por meio de uma memória de operandos compartilhada. Deve-se utilizar no programa do CP uma ou mais chamadas a uma função de comunicação com o Processador Multitarefa AL-2005 (módulo F-2005.016). Este módulo função permite que o AL-2005 obtenha acesso a memória imagem dos operandos do CP, podendo realizar a leitura ou escrita dos operandos do CP.

Através da conexão de um microcomputador padrão IBM-PC® ao Processador Multitarefa AL-2005 pode-se realizar carga e depuração de programas aplicativos. Para tanto deve-se utilizar o carregador AL-3860, fornecido no CD que acompanha o produto, e o canal serial dedicado para carga de programa no Processador Multitarefa AL-2005.

Aplicações do Processador Multitarefa AL-2005

O Processador Multitarefa AL-2005/RTMP pode ser utilizado para realizar tarefas de alta complexidade liberando o processador principal do CP para realizar suas tarefas convencionais, tais como varredura e acionamento, permitindo uma distribuição mais equilibrada da carga de processamento entre os dois processadores.

O processador AL-2005 suporta a escrita de aplicações seqüenciais tradicionais ou aplicações multitarefa. Esta última permite um maior aproveitamento do tempo do processador, uma vez que qualquer tarefa que não necessita da UCP em um determinado momento, pode liberá-la para outra tarefa ou aplicação.

Este manual inclui a descrição do ambiente de desenvolvimento para aplicações no AL-2005, bem como fornece uma referência A-Z das funções da biblioteca do sistema.

O programa executivo do AL-2005, também chamado de BIOS, é um sistema operacional multitarefa de tempo real preemptivo, que suporta a execução simultânea de vários programas aplicativos. Para tanto, torna disponível ao projetista de aplicações, uma biblioteca de funções.

Através do suporte desta biblioteca de funções é possível:

- desenvolver aplicativos multitarefa utilizando funções de criação de tarefas, espera de eventos, espera de recursos, semáforos, identificação de tarefas etc.

- utilizar aritmética de ponto flutuante.
- realizar alocação dinâmica de memória.
- acessar os operandos do CP, através de funções de escrita e leitura a operandos simples e tabelas.
- acessar as placas seriais conjugadas ao hardware do AL-2005, nos padrão RS-485 ou RS-232C.

Documentos Relacionados a este Manual

Para obter informações adicionais sobre o processador AL-2005 podem ser consultados outros documentos (manuais e características técnicas) além deste. Estes documentos encontram-se disponíveis em sua última revisão em www.altus.com.br.

Cada produto possui um documento denominado Característica Técnica (CT), onde encontram-se as características do produto em questão. Adicionalmente o produto pode possuir Manuais de Utilização (o código do manuais são citados na CT).

Aconselha-se os seguintes documentos como fonte de informação adicional:

- Manual de Utilização do AL-2003.
- Manual de Utilização do AL-2004.
- Manual de Programação do MT4100.

Inspeção Visual

Antes de proceder à instalação, é recomendável fazer uma inspeção visual cuidadosa dos equipamentos, verificando se não há danos causados pelo transporte. Verifique se todos os componentes de seu pedido estão em perfeito estado. Em caso de defeitos, informe a companhia transportadora e o representante ou distribuidor Altus mais próximo.

CUIDADO:
Antes de retirar os módulos da embalagem, é importante descarregar eventuais potenciais estáticos acumulados no corpo. Para isso, toque (com as mãos nuas) em uma superfície metálica aterrada qualquer antes de manipular os módulos. Tal procedimento garante que os níveis de eletricidade estática suportados pelo módulo não serão ultrapassados.

É importante registrar o número de série de cada equipamento recebido, bem como as revisões de software, caso existentes. Essas informações serão necessárias caso se necessite contatar o Suporte Técnico da Altus.

Suporte Técnico

Para entrar em contato com o Suporte Técnico da Altus em São Leopoldo, RS, ligue para +55-51-3589-9500. Para conhecer os centros de Suporte Técnico da Altus existentes em outras localidades, consulte nosso site (www.altus.com.br) ou envie um email para altus@altus.com.br.

Se o equipamento já estiver instalado, tenha em mãos as seguintes informações ao solicitar assistência:

- os modelos dos equipamentos utilizados e a configuração do sistema instalado
- o número de série da UCP
- a revisão do equipamento e a versão do software executivo, constantes na etiqueta afixada na lateral do produto
- informações sobre o modo de operação da UCP, obtidas através do programador MasterTool
- o conteúdo do programa aplicativo (módulos), obtido através do programador MasterTool
- a versão do programador utilizado

Mensagens de Advertência Utilizadas neste Manual

Neste manual, as mensagens de advertência apresentarão os seguintes formatos e significados:

PERIGO:
Relatam causas potenciais, que se não observadas, *levam* a danos à integridade física e saúde, patrimônio, meio ambiente e perda da produção.

CUIDADO:
Relatam detalhes de configuração, aplicação e instalação que *devem* ser seguidos para evitar condições que possam levar a falha do sistema e suas consequências relacionadas.

ATENÇÃO:
Indicam detalhes importantes de configuração, aplicação ou instalação para obtenção da máxima performance operacional do sistema.

2. Descrição Técnica

Este capítulo apresenta as características técnicas do produto AL-2005, abordando as partes integrantes do sistema, sua arquitetura, características gerais e elétricas.

Descrição do Painel

A figura 2-1 mostra o painel do produto AL-2005.

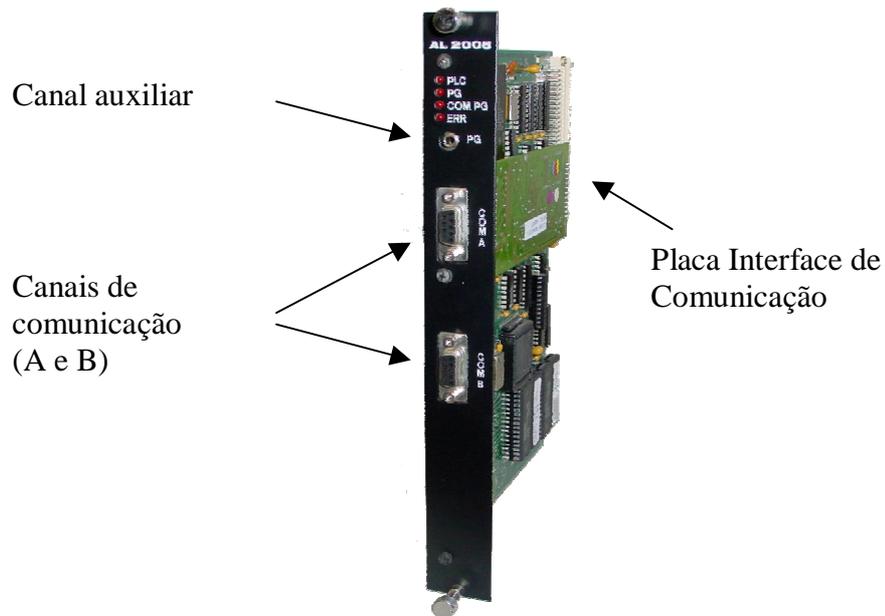



Figura 2-1: Painel do Processador AL-2005

Características Técnicas

Características Gerais

	AL-2005
Tipo de módulo	Real Time Multitasking Processor
Processador	80C186 20 MHz 16 bits
Memória RAM	256 Kb (64 Kb ocupados pela BIOS)
Memória Flash para aplicativos	256 Kb (64 Kb ocupados pela BIOS)
Carga de programas aplicativos no processador	Via canal serial através do carregador AL-3860
Sistema operacional	Multitarefa preemptivo
Linguagem de Programação dos aplicativos	Linguagem "C"
Comunicação com o CP	Através de módulo F chamado por meio de uma instrução CHF
Acesso aos operandos do CP	Via memória imagem
Barramentos de dados internos	16 bits
Velocidade canal serial	Com apenas um canal serial : 38400 bauds Com dois canais: 19200 bauds
Tempo máximo para desligamento do sinal de RTS	Com uma placa AL-2405: 150uS Com duas placas AL-2405: 950uS
Número máximo de tarefas em cada AL-2005	8
Taxa de transferência de dados com memória imagem da UCP	Melhor que 1 Mbyte/s
Portas Seriais COM1 e COM2	RS232C ou RS-485 conforme interface de comunicação AL-2405
Consumo	850 mA @ 5 Vcc considerar consumo de placas adicionais conectadas ao Processador AL-2005
Nível de severidade de descargas eletrostáticas (ESD)	Conforme a norma IEC 1131 nível 3
Imunidade a ruído elétrico tipo onda oscilatória	Conforme norma IEC1131, nível de severidade A e IEEE C37.90.1 (SWC)
Proteção contra choque elétrico	Conforme norma IEC-536 (1976), classe I, quando instalado em bastidor
Peso	Sem embalagem: 500 g Com embalagem: 700 g
Umidade relativa do ar	5 a 95% sem condensação
Indicação de estado	4 LEDs indicadores de estado
Temperatura máxima de operação	60 ^{o}C
Temperatura de armazenagem	-25 a 70 °C

Tabela 2-1 Características Técnicas Gerais

Arquitetura do Processador Multitarefa AL-2005

Um sistema de tempo real se caracteriza pela necessidade de responder adequadamente a eventos ocorridos assincronamente no tempo. Um sistema multitarefa é um conjunto de atividades ou tarefas que podem ser realizados sem interferência em outros processos. Um sistema com várias tarefas que devem ser executadas simultaneamente e com um tempo de execução crítico é chamado de "real-time multitasking system".

O processador AL-2005 implementa um sistema operacional multitarefa, o que facilita a solução de sistemas de tempo real. O sistema operacional do AL-2005 utiliza um método de "time-slice" (método de tempo compartilhado) para a execução de aplicativos independentes uns dos outros.

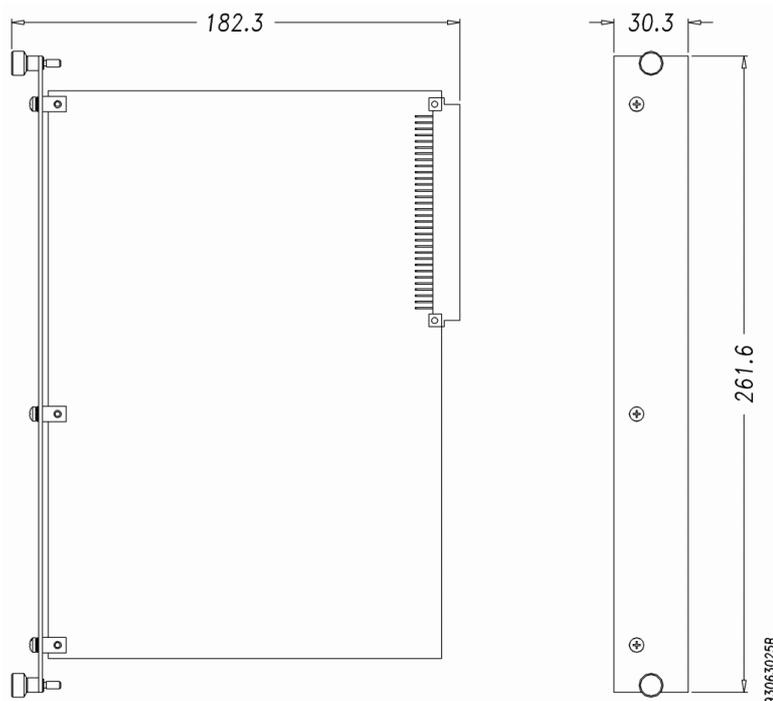
Cada aplicativo pode possuir um conjunto de tarefas. O chaveamento de tarefas segue o método preemptivo (prioridade de tarefa). Portanto o sistema operacional do processador AL-2005 segue dois métodos:

- Time-slice: tempo compartilhado para cada aplicação residente no AL-2005.
- Preemptivo: para cada tarefa de um determinado aplicativo deve-se atribuir uma prioridade de execução. As tarefas podem ser interrompidas por uma de maior prioridade.

A arquitetura do processador AL-2005 é baseada no microprocessador de 16 bits Intel® 80C186, que possui elevada integração de recursos de hardware e é especialmente indicado para aplicações industriais. O código das instruções deste microprocessador é compatível com o dos demais microprocessadores da Intel® utilizados nos computadores IBM PC® (80386, 80486, etc.).

Características do Equipamento

Dimensões Físicas

Dimensões em milímetros.

Dados para Compra

Itens Integrantes

A embalagem do produto contém um

- Módulo AL-2005/RTMP: processador AL-2005/RTMP (Real Time Multitasking Processor)
- CD Card com os seguintes itens:
    - Módulo função de interface UCP/AL-2005: F-2005.016.
    - Carregador AL-3860.
    - Biblioteca de Funções.
    - Programas Exemplos: DEMOS.
    - Manual de Utilização.
    - Tutorial.

Código do Produto

O seguinte código deve ser usado para compra do produto:

Código	Denominação
AL-2005	Real Time Multitask Processor

Produtos Relacionados

Os seguintes produtos devem ser adquiridos separadamente quando necessário:

Código	Denominação
AL-2405/485I	Módulo serial RS-485 isolado
AL-2405/232	Módulo serial RS-232C
AL-1340	Cabo DB25 P2 Estéreo
AL-1327 ou AL-1715	Cabo DB9-RJ45 PC/AL-2005
AL-2302	Cabo DB9 macho para rede RS-485
MT4100	Programador MasterTool

ATENÇÃO:
Para o desenvolvimento de programas aplicativos para o processador AL-2005 é obrigatória a utilização do compilador Borland C++ versão 3.1, nos modelos de memória LARGE ou SMALL. Além do compilador, é necessário também o utilitário de relocação LOCATE da Paradigm, versão 4.0.

ATENÇÃO:
Para aquisição deste compilador, deve ser adquirida a versão 5.02 ou posterior do produto BorlandC Compiler junto aos representantes da Borland e requisitada cópia do compilador Borland C ++ 3.1.

3. Configuração

O Processador Multitarefa AL-2005 é um módulo que opera em conjunto com a UCP AL-2003 ou AL-2004. Desta forma, o Processador Multitarefa AL-2005 deve ser declarado como um módulo no barramento da UCP.

Esta declaração é feita através do programador MasterTool. A figura a seguir mostra a tela de declaração dos módulos no barramento da UCP:

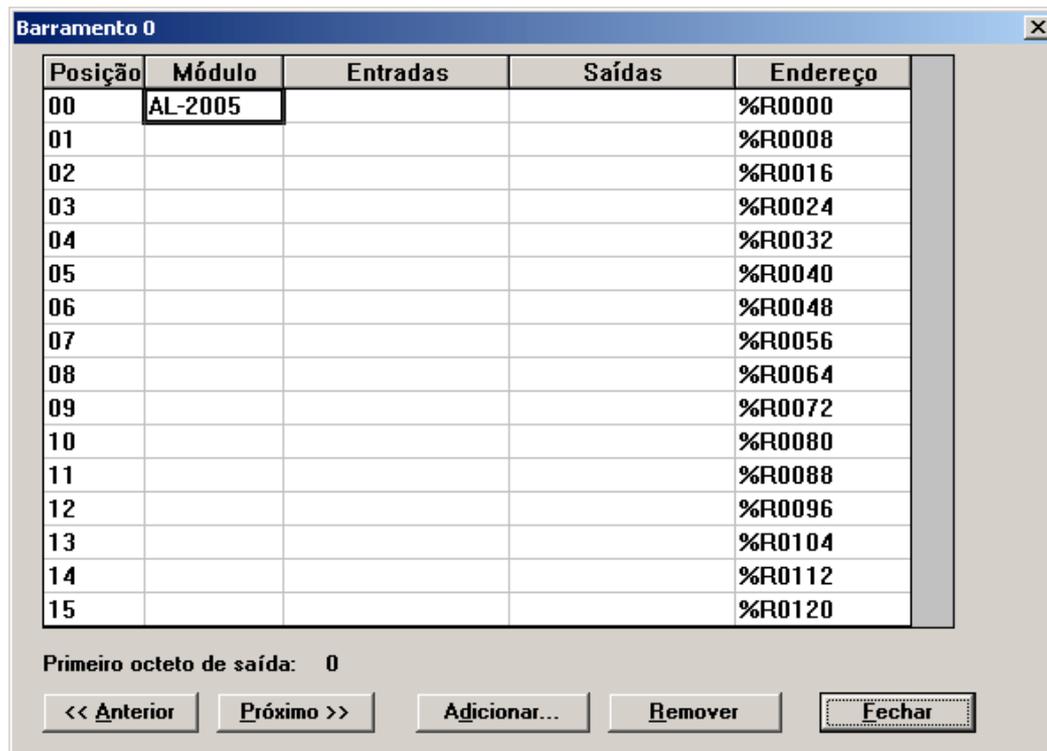



Figura 3-1 Tela de Configuração do Barramento no Programador MasterTool

Além da declaração do AL-2005 no barramento, é necessário a utilização de um módulo função – F-2005.016 - no programa aplicativo do CP. Este módulo é responsável pela comunicação entre o processador e a UCP.

O processador AL-2005 é baseado em microprocessador 80186 da Intel, permitindo que programas aplicativos sejam desenvolvidos utilizando compiladores para microcomputador padrão IBM PC, que geram código compatível com o do processador.

No entanto, por questões de implementação de seu programa executivo, o desenvolvimento de programas aplicativos para o AL-2005 está limitado à utilização do compilador Borland C++ versão 3.1.

Ambiente de Desenvolvimento de Aplicações para o AL-2005

O desenvolvimento de aplicações pelo usuário para o AL-2005 deve ser feito a partir de uma das aplicações exemplos, identificadas pelo arquivo de projeto DEMO.PRJ, fornecidas com o produto AL-2005.

Uma característica importante fornecida pela BIOS do AL-2005 é a capacidade de depuração simbólica remota, o que permite o uso do Turbo Debbuger da Borland para a depuração de programas aplicativos.

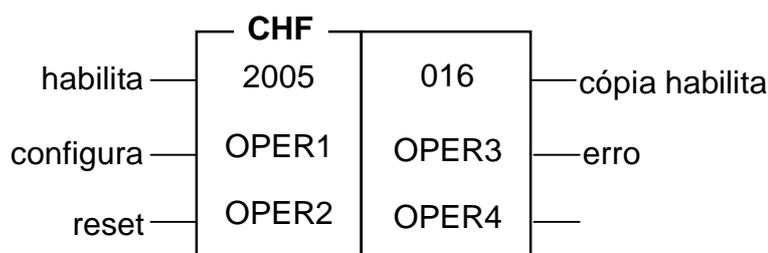
As aplicações desenvolvidas podem ser verificadas tomando como exemplo a aplicação exemplo depurável identificada pelo arquivo de projeto DEMOTD.PRJ.

Para que o processador AL-2005 obtenha acesso à memória imagem do CP (memória de operandos) é necessária a chamada da função de comunicação F-2005.016 no programa do CP.

O programa aplicativo responsável por atuar na janela de tempo disponibilizada pela F-2005.016 deve ser desenvolvido e carregado no processador AL-2005. A carga de aplicativos no processador AL-2005 é efetuada pelo usuário pela interface serial RS-232C de um computador PC compatível, com o uso do programa carregador AL-3860.

F-2005.016 - Função de Comunicação CP com o AL-2005

A comunicação entre o processador AL-2005 e a UCP é efetuada através de uma área de memória compartilhada por ambos, à qual cada um dos módulos tem acesso de forma exclusiva. A iniciativa da comunicação é sempre comandada pela chamada do módulo de função F-2005.016 dentro do programa aplicativo do CP, fazendo com que, a partir deste instante, o processador AL-2005 disponha de uma janela de tempo de até 3200 µs para acessar as áreas de comunicação com o CP.

O módulo de função F-2005.016 implementa a comunicação entre a UCP e o processador AL-2005. É também usado para transferir informações de configuração do CP para os programas aplicativos do AL-2005, através de tabelas de memória. Estas informações de configuração podem, por exemplo, estabelecer um mapeamento, ou seja, relações, entre operandos do CP e variáveis/parâmetros de algum dispositivo externo, conectado através dos canais seriais do processador AL-2005.

As entradas da instrução de chamada da F-2005.016 são:

- **habilita:** quando acionada, a função está habilitada a executar, abrindo uma janela de tempo para acesso do processador AL-2005 às áreas de comunicação com o CP ou possibilitando a transferência de informações de configuração.
- **configura:** quando acionada, novas informações de configuração são transferidas do CP para o processador AL-2005. A entrada **configura**, após ativada, deve ser desligada quando a saída de **erro** estiver no estado desligado.
- **reset: q**uando acionada, faz com que se perca a configuração corrente. A entrada **reset**, após ativada, deve ser desligada quando a saída de **erro** estiver no estado desligado.

As saídas da instrução de chamada da F-2005.016 são:

- **cópia habilita: é** uma cópia da entrada **habilita.**
- **erro: e**sta saída é energizada quando a comunicação entre o CP e o processador AL-2005 não se realiza a contento: ou porque o processador AL-2005 não aproveita a janela de tempo concedida pelo CP, ou porque os operandos da função foram mal especificados quanto a seus tipos ou endereços.

As células da instrução CHF utilizada para a chamada da F-2005.016 são programadas do seguinte modo:

- **OPER1:** número de parâmetros passados para a função em **OPER3**: deve ser obrigatoriamente uma constante memória com o valor 3 (KM+00003).
- **OPER2:** número de parâmetros passados em **OPER4**: KM+00000, já que **OPER4** não é utilizado.
- **OPER3:** parâmetros passados para a função, declarados através de uma janela visualizada no programador de CPs MasterTool quando a instrução CHF é editada; o número de parâmetros editáveis é especificado em **OPER1**, sendo fixado em 3 para este módulo:
  RXXXX: endereço da placa AL-2005 no barramento do CP . Consulte documentação do bastidor em uso para definição de endereços válidos para módulos inteligentes.
- **OPER4:** não utilizado

O Carregador de Aplicativos AL-3860

O carregador AL-3860 permite a carga de aplicativos para o processador AL-2005, bem como a execução de operações de manutenção, tais como leitura de aplicativos já carregados e remoção dos mesmos.

Para a execução do carregador AL-3860, primeiramente, é necessário conectar-se o cabo AL-1327 na porta de comunicação serial desejada e ligar o AL-2005. O AL-2005 indicará que não existe aplicação carregada, utilizando-o pela primeira vez. Após, é necessário selecionar a porta de comunicação desejada, pressionando-se o botão Porta Serial. Finalmente, executar o programa AL-3860 no diretório corrente.

A tela de apresentação do carregador AL-3860 é mostrada na figura a seguir:

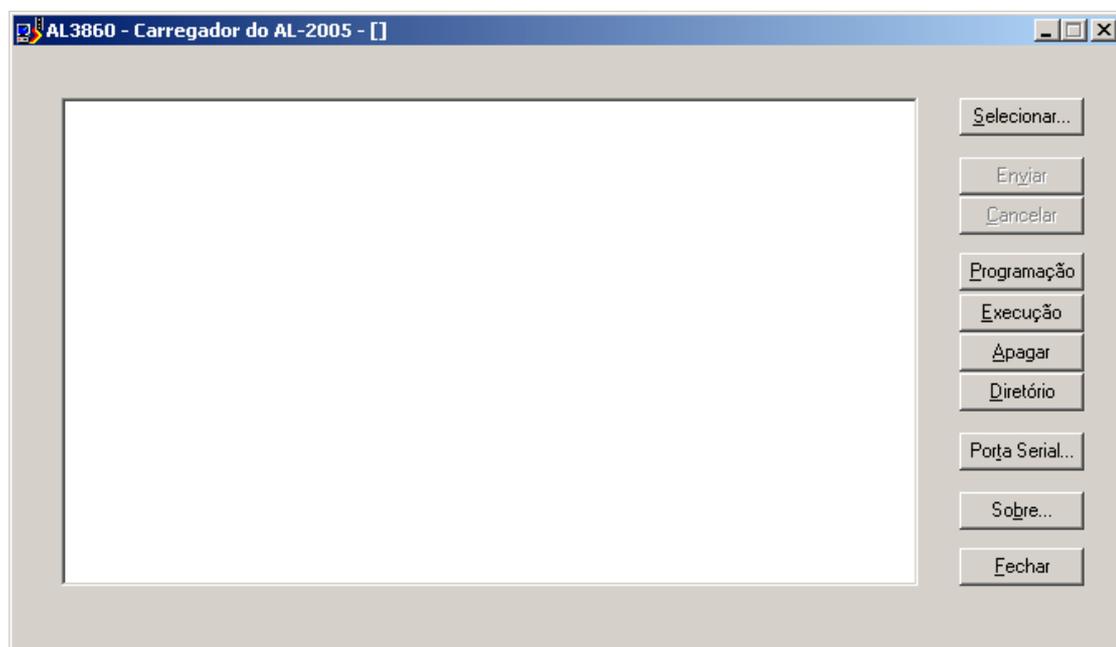



Figura 3-2 Carregador AL-3860

ATENÇÃO:
Para executar as operações no AL-2005, o mesmo deve encontrar-se, **obrigatoriamente**, no estado programação.

A área branca é destinada a mostrar informações sobre as aplicações carregadas no AL-2005, bem como mensagens orientativas, auxiliando a utilização do software.

As seguintes operações estão disponíveis no AL-3860 através de botões de comando:

Botão Selecionar

Este comando permite a seleção de arquivos de programas aplicativos executáveis (*.EXE) do disco, para posterior envio ao AL-2005. Nenhum outro tipo de arquivo pode ser selecionado por esta opção. Ao selecionar um arquivo o carregador AL-3860 exibe o seu nome na barra de título, conforme mostrado na figura a seguir:

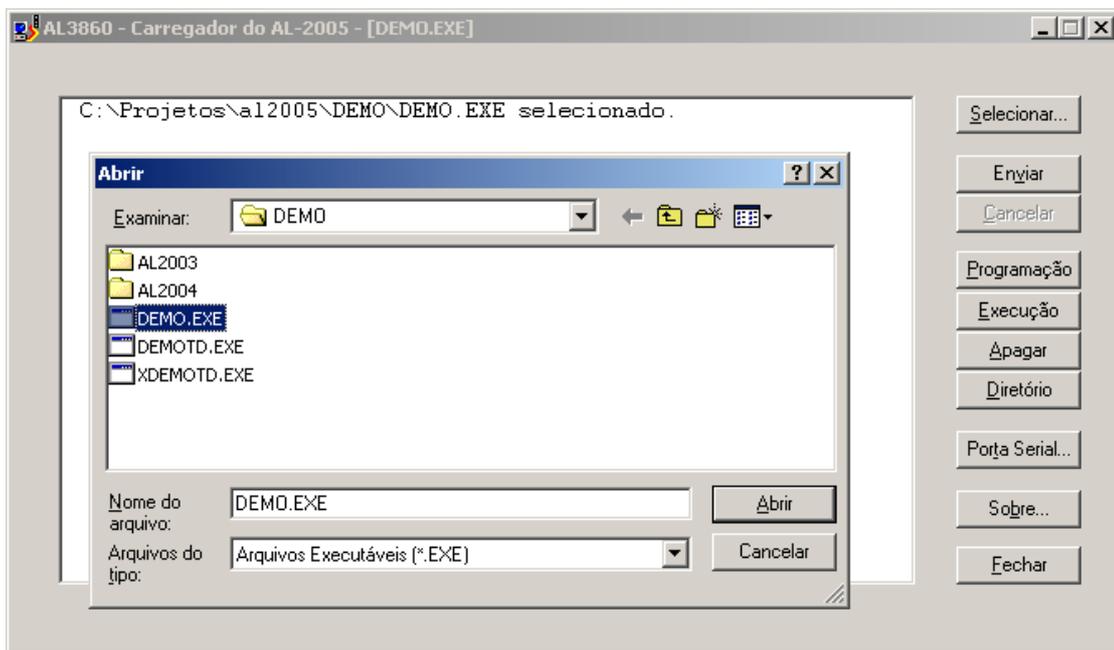



Figura 3-3 Seleção de Arquivo para Carga

Botão Enviar

Esta opção envia o programa aplicativo para a memória do processador AL-2005. Antes da efetiva transmissão do arquivo o AL-3860 verifica se o programa aplicativo selecionado é um arquivo executável no AL-2005 e se existe espaço de memória de código e de dados para a carga.

Botão Cancelar

Esta opção cancela a transmissão da aplicação para o AL-2005.

Botão Programação

Este comando passa o processador para o estado programação. É no estado programação que são permitidas as operações de leitura de diretório do processador AL-2005, exclusão de aplicativos da memória e envio de programas aplicativos.

Botão Execução

Este comando passa o AL-2005 para o estado execução. Ao receber este comando, o processador verifica se existe algum aplicativo carregado em sua memória. Caso haja algum aplicativo na memória, a BIOS dispara a sua execução. Caso contrário, o processador entra em estado de erro de programa.

Botão Apagar

Esta opção apaga todos os aplicativos residentes na memória do processador AL-2005.

ATENÇÃO:
O processo de apagamento dos programas aplicativos residentes na memória do processador AL-2005 é um processo irreversível.

Botão Diretório

Exibe o diretório de aplicativos carregados na memória Flash do processador AL-2005. Para cada aplicativo carregado no processador é exibido o número da aplicação, bem como o número de bytes utilizados para código (na memória Flash) e o número de bytes utilizados para dados (na memória RAM). A figura a seguir apresenta uma tela exemplo de um diretório.

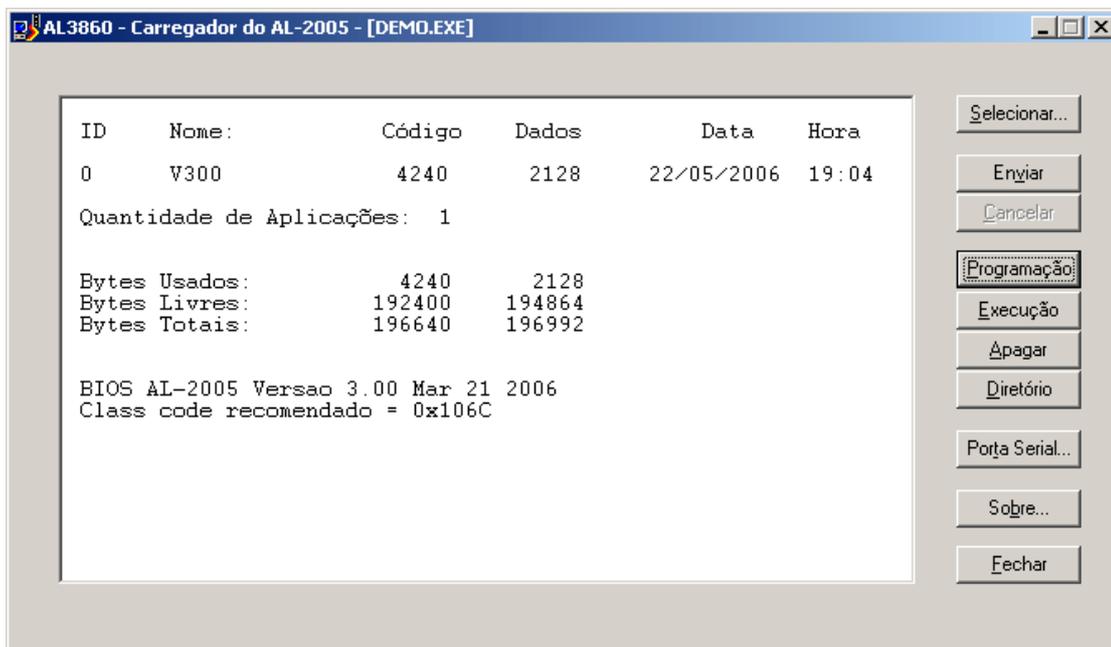



Figura 3-4 Tela do diretório

Botão Porta Serial

Esta opção permite selecionar o canal serial do microcomputador que será utilizado na comunicação com o processador AL-2005. A figura a seguir mostra a janela de seleção do canal.

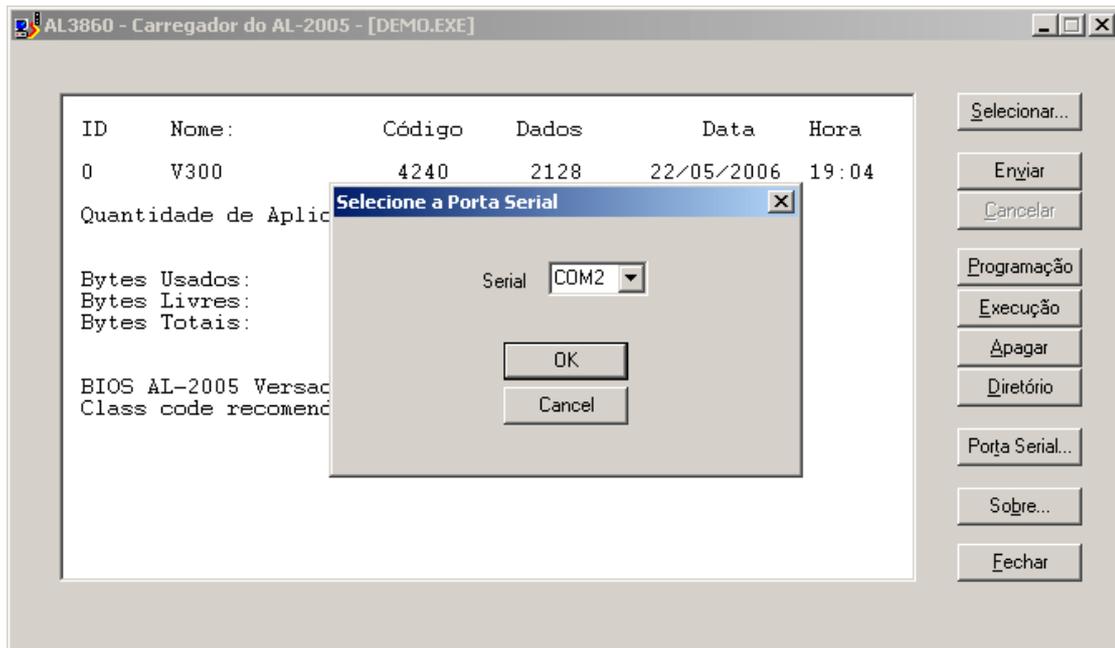



Figura 3-3 Seleção do Canal Serial

Botão Sobre

Esta opção mostra a versão do AL-3860.

Botão Fechar

Permite finalizar a execução do carregador AL-3860.

4. Sistema Operacional

Um sistema de tempo real é caracterizado pela necessidade de uma resposta rápida a eventos que ocorrem assincronamente no tempo. Um sistema multitarefa é aquele no qual várias atividades ou processos devem ser realizados simultaneamente sem interferência entre si.

Um sistema no qual várias atividades devam operar simultaneamente em tempos críticos é chamado de sistema multitarefa em tempo real.

O sistema operacional do processador AL-2005 fornece uma solução simples para a complexidade de múltiplas tarefas sendo executadas em tempo real, supervisionando a execução ordenada deste conjunto de tarefas.

Uma tarefa nada mais é do que um módulo de programa. Um programa aplicativo é composto por uma ou mais tarefas, onde cada tarefa pode resolver um problema específico e fornecer uma determinada capacidade funcional ao programa.

Este capítulo apresenta o sistema operacional do processador AL-2005, descrevendo como devem ser criadas e se relacionam entre si as diferentes tarefas carregadas.

Escalonador de Tarefas

O escalonador de tarefas procura em uma lista de tarefas disponíveis aquela com a maior prioridade e em condições de execução. A prioridade da execução das tarefas é determinada pelo projetista das mesmas, exceto no caso dos próprios programas aplicativos que possuem um valor de prioridade pré-definido. Se não houver nenhuma tarefa pronta para ser executada, o sistema operacional, com as interrupções habilitadas, aguarda que algum evento externo gere uma interrupção.

O sistema operacional inicia a execução de uma tarefa pelo endereço de início da mesma, também definido pelo projetista. A tarefa é executada como se fosse o único programa no sistema.

Assim que inicia sua execução, a tarefa parece operar sem interrupções. As interrupções que porventura possam ocorrer são tratadas pelo sistema operacional de uma forma transparente para a tarefa. Ao entrar em execução, a tarefa inibe a execução de qualquer outra tarefa com prioridade menor do que a sua, permanecendo neste estado até que decida renunciar ao controle, mesmo que temporariamente, através da chamada de uma função do sistema operacional.

A tarefa termina ao retornar para o escalonador do sistema operacional que volta a procurar a próxima tarefa de maior prioridade pronta a executar, dando-lhe o controle do processador.

O sistema operacional oferece um conjunto de serviços que podem ser invocados pela tarefa através de chamadas de funções, que constituem a chamada biblioteca de suporte do sistema. Uma tarefa pode, por exemplo, enviar uma mensagem para outra tarefa, aguardar por um evento ou por um intervalo de tempo. Se a tarefa desejar aguardar um evento, o serviço correspondente do sistema operacional suspenderá a tarefa e requisitará ao escalonador a execução da próxima tarefa com maior prioridade pronta a executar.

O sistema operacional atua como chaveador de contextos, supervisionando a execução ordenada das tarefas. Emprega um algoritmo de escalonamento preemptivo e comandado por prioridade que assegura que a tarefa de maior prioridade que esteja pronta para executar um trabalho útil sempre terá o controle do processador. O sistema fará o chaveamento de tarefas se receber uma requisição da tarefa em execução para realizar uma operação que invoca uma tarefa de maior prioridade. A tarefa em execução pode, por exemplo, requisitar ao sistema operacional o disparo de uma tarefa de maior prioridade através do envio de uma mensagem para a mesma.

Os serviços do sistema operacional diretamente relacionados com o controle de tarefas são os seguintes:

Controle de tarefas	ChangeTaskPriority	Altera a prioridade de uma tarefa
	CreateTask	Cria uma nova tarefa
	DeleteTask	Remove uma tarefa do sistema operacional
	EndTask	Termina a execução da tarefa corrente
	GetTaskDescriptor	Obtém o ponteiro para o descritor de uma tarefa
	GetTaskID	Obtém o identificador da tarefa corrente
	GetTaskStatus	Obtém o estado de uma tarefa
	KillTask	Mata uma tarefa
	LocateTask	Localiza uma tarefa através de seu nome
	ResetPendingWake	Reseta uma requisição pendente para acordar a tarefa corrente
	ResetTaskWake	Reseta uma requisição pendente para acordar uma tarefa
	StartTask	Dispara a execução de uma tarefa
	StopTask	Para (termina) a execução de uma tarefa
	Wait	Aguarda incondicionalmente até ser acordada
	WaitTime	Aguarda um intervalo de tempo (“delay”)
	WakeCallingTask	Acorda a tarefa que enviou uma mensagem para a tarefa corrente
	WakeTask	Acorda uma tarefa no estado de espera

Criação de uma Tarefa

As tarefas podem ser pré-definidas, como no caso de um programa aplicativo carregado no processador AL-2005, ou podem ser criadas dinamicamente por outras tarefas durante sua execução.

O sistema operacional AMX do AL-2005 suporta no máximo 10 tarefas simultâneas, sendo que destas, 2 tarefas são reservadas para a própria BIOS do AL-2005: a primeira tarefa é a própria aplicação (main) da BIOS, e a segunda tarefa é a de comunicação com o carregador AL-3860. As outras 8 tarefas estão disponíveis para serem utilizadas pelas aplicações a serem carregadas no AL-2005.

Cada aplicação carregada no AL-2005 é considerada também como uma nova tarefa.

Exemplo: se forem carregadas duas aplicações no AL-2005, e onde cada uma delas ainda crie mais duas tarefas (CreateTask), teremos 8 tarefas executando, das 10 disponíveis:

- tarefa 1: BIOS.
- tarefa 2: tarefa criada pela “main” da BIOS (int de comunicação).
- tarefa 3: “main” da primeira aplicação carregada.
- tarefa 4: “main” da segunda aplicação carregada.
- tarefa 5: tarefa 1/2 criada pela “main” da primeira aplicação carregada.
- tarefa 6: tarefa 2/2 criada pela “main” da primeira aplicação carregada.
- tarefa 7: tarefa 1/2 criada pela “main” da segunda aplicação carregada.
- tarefa 8: tarefa 2/2 criada pela “main” da segunda aplicação carregada.
- tarefa 9: disponível.
- tarefa 10: disponível.

A criação dinâmica de uma tarefa requer a inicialização de uma estrutura de dados de descrição da tarefa, que contém informações relevantes para que o sistema operacional controle a execução da mesma.

Esta estrutura encontra-se definida no arquivo AL2005.H da seguinte forma:

```
typedef struct
{
  void        far (far *TaskProcedure) ();
  char        Tag1;
  char        Tag2;
  char        Tag3;
  char        Tag4;
  unsigned int far *lpnStack;
  unsigned int nStackSize;
  unsigned int nTaskAttributes;
  unsigned int nPriority;
  unsigned int nTimeSlice;
  int         nMail0Size;
  int         nMail1Size;
  int         nMail2Size;
  int         nMail3Size;
} TASKDESCR;
```

onde:

TaskProcedure é um ponteiro para a função que será chamada pelo sistema operacional quando a tarefa for disparada.

ATENÇÃO: Toda função usada para disparar uma nova tarefa deve ser declarada como HUGE. O motivo disto é garantir que o registrador de segmento de dados (DS) da tarefa seja preservado quando houver chaveamento de tarefas.

Tag1 a Tag4 são quatro bytes usados para definir um tag único, isto é, um nome, para cada tarefa; normalmente utilizam-se quatro caracteres ASCII como tag da tarefa. A função *LocateTask* pode ser usada para encontrar o identificador de uma tarefa através de seu tag.

lpnStack é o endereço da pilha da tarefa. Para cada tarefa deve ser alocada uma área em RAM para a sua pilha. Deve ser passado o endereço do fim da pilha. A partir da versão 2.24 da BIOS, este endereço passou a ser ajustado automaticamente para permitir que a tarefa execute cálculos em ponto flutuante sem afetar outras áreas.

nStackSize é o tamanho da pilha da tarefa em bytes. Deve ser um número par de bytes. O tamanho da pilha de cada tarefa é dependente da aplicação. Deve ter no mínimo 64 palavras (128 bytes) para permitir com que a tarefa receba mensagens e seja interrompida e suspensa por tarefas de maior prioridade. Além disto, deve-se alocar espaço suficiente para satisfazer as necessidades particulares de cada tarefa.

nTaskAttributes deve ser sempre zero.

nPriority é a prioridade de execução da tarefa. A prioridade das tarefas varia de 1 (maior) a 127 (menor). Mais de uma tarefa podem ter a mesma prioridade.

ntimeSlice deve ser sempre zero.

nMail0Size a *nMail3Size* definem o número máximo de envelopes de mensagens que podem estar em cada uma das quatro caixas postais que uma tarefa pode possuir. A caixa postal 0 é a de maior prioridade e a 3 a de menor prioridade.

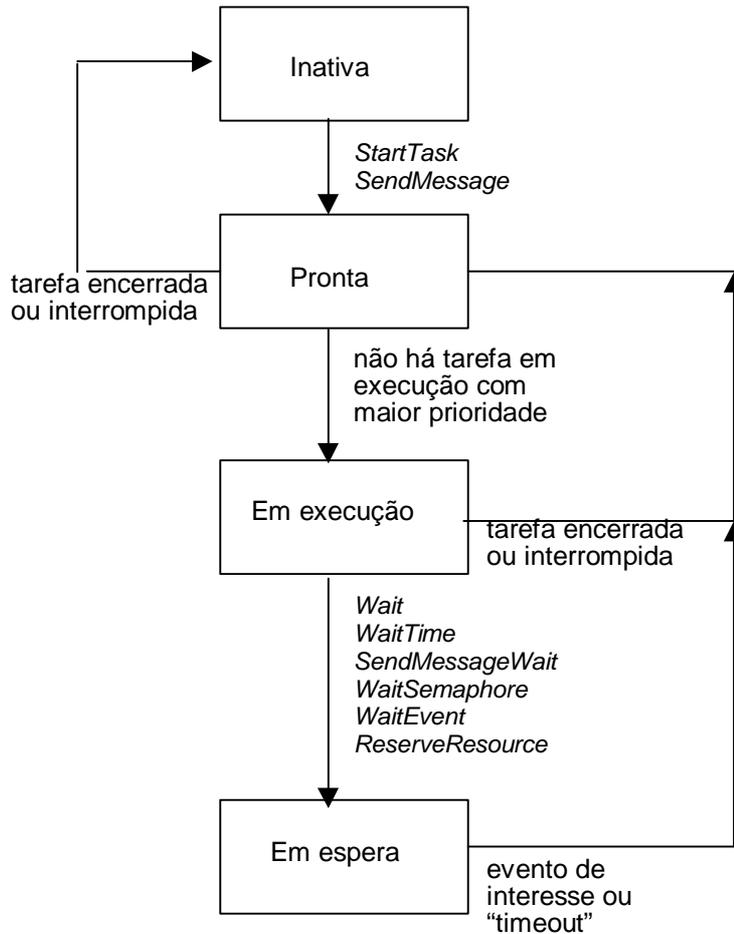
Um exemplo de definição do descritor de uma tarefa aparece a seguir:

```
#define PRIOR_TASK              64      /* Prioridade da tarefa              */
#define TAMANHO_PILHA           1200    /* Tamanho da pilha, múltiplo de 16  */
unsigned int Pilha [TAMANHO_PILHA];     /* Pilha da tarefa                   */
void huge TrataAlnetI (void);           /* Tarefa                            */
static TASKDESCR DescrAlnetI = {        /* Descrição da estrutura            */
    TrataAlnetI,                        /* Ponto de entrada                  */
    'A', 'L', 'N', 'I',                 /* Nome da tarefa (4 caractere)      */
    Pilha + TAMANHO_PILHA,              /* Ponteiro para o fim da pilha      */
    TAMANHO_PILHA * sizeof (int),       /* Tamanho da pilha em bytes         */
    0,                                  /* Atributos da tarefa               */
    PRIOR_TASK                          /* Prioridade da tarefa              */
    0,                                  /* Tempo (sempre zero)               */
    0, 0, 0, 0                          /* Tamanho das caixas postais        */
    };
```

Quando uma tarefa é criada dinamicamente, o sistema operacional retorna o identificador da tarefa criada para o criador. É responsabilidade da aplicação manter este identificador, de modo a poder referir-se à tarefa criada subseqüentemente.

Estados de uma Tarefa

Uma tarefa sempre se encontrará em um dos seguintes estados:

- *Inativa*: é o estado da tarefa quando a mesma é criada. Neste estado a tarefa não possui nenhuma requisição pendente a ser atendida.
- *Pronta*: uma tarefa encontra-se no estado de pronta quando tiver alguma requisição pendente a atender ou estiver pronta a retomar sua execução após ter sido interrompida ou estar aguardando.
- *Em execução*: a tarefa está sendo executada pelo processador. Só é possível uma única tarefa encontrar-se neste estado a cada instante.
- *Em espera*: a tarefa encontra-se bloqueada, aguardando a ocorrência de algum evento.

Início de uma Tarefa

Os programas aplicativos carregados no processador AL-2005 são tarefas pré-definidas. Logo que o sistema operacional é iniciado, na energização do processador, a memória de usuário é varrida em busca de programas aplicativos. Quando um programa aplicativo é encontrado, o sistema operacional cria uma tarefa para este aplicativo e em seguida utiliza a função *StartTask* para disparar sua execução.

Quando o programa aplicativo inicia sua execução, o mesmo pode criar novas tarefas através da função *CreateTask*, como explicado na seção anterior.

Qualquer tarefa em execução pode requisitar a execução de outra tarefa do sistema, utilizando uma entre três funções:

StartTask dispara a execução da outra tarefa sem passar nenhuma mensagem

SendMessage envia uma mensagem para a outra tarefa

SendMessageWait envia uma mensagem para a outra tarefa e aguarda que a mesma seja recebida e processada

Tarefas que não recebem mensagens e que são portanto iniciadas chamando-se *StartTask*, devem ser escritas como uma função sem parâmetros formais.

Por exemplo, uma tarefa que se encerra imediatamente seria:

```
void far task1 (void)
{
}
```

Tarefas que devem receber uma mensagem (disparadas por *SendMessage* ou *SendMessageWait*) são escritas como funções que recebem parâmetros. Por exemplo, uma função que recebe um inteiro como mensagem e então encerra apareceria como:

```
void far task2 (int message)
{
}
```

As chamadas das funções *SendMessage* e *SendMessageWait* passam como parâmetro uma mensagem que é movida para um envelope. Este envelope é inserido então na caixa postal da tarefa destino, no nível de prioridade indicado pela chamada.

Uma tarefa que executa uma chamada à função *SendMessageWait* suspende a si mesma (entra em estado de espera) até que a tarefa sendo chamada receba a mensagem e responda à chamada.

Prioridade

As prioridades das tarefas são usadas pelo escalonador do sistema operacional para determinar a tarefa a ser executada. A qualquer instante sempre estará em execução a tarefa com a maior prioridade que estiver capacitada a executar.

As prioridades das tarefas vão de 1 (maior) a 127 (menor). A prioridade dos programas aplicativos carregados no processador AL-2005 é pré-definida em 64. A prioridade das tarefas criadas dinamicamente é definida na sua criação, através do campo *nPriority* do descritor da tarefa *TASKDESCR.*

Se mais de uma tarefa possuir a mesma prioridade, o sistema operacional irá atribuir prioridades relativas às diferentes tarefas, de acordo com a ordem cronológica com que foram criadas, com a primeira tarefa criada tendo a maior prioridade.

Uma tarefa pode alterar sua prioridade ou a prioridade de qualquer outra tarefa através da função *ChangeTaskPriority*, mas esta prática não é recomendada.

Execução de uma Tarefa

O sistema operacional dispara uma tarefa fazendo uma chamada FAR para o endereço da função especificada no descritor da tarefa quando a mesma é criada, no campo *TaskProcedure*. A tarefa somente será iniciada se não houver nenhuma outra tarefa de maior prioridade em condições de ser executada. Portanto uma tarefa somente entra em execução se todas as outras tarefas de maior prioridade estiverem inativas ou suspensas por algum motivo.

Quando uma tarefa inicia sua execução, tem completo controle sobre o processador. O sistema de interrupções deve permanecer habilitado a fim de permitir o atendimento das interrupções. Se a tarefa necessitar desabilitar as interrupções por algum motivo, recomenda-se que isto seja feito no menor período de tempo possível a fim de não degradar o tempo de resposta das interrupções.

Uma tarefa pode executar sem se preocupar com o fato de que ocorrerão interrupções e que as mesmas serão atendidas. Se tarefas de maior prioridade ficarem prontas para execução, a tarefa será suspensa temporariamente pelo sistema operacional. Quando a tarefa de maior prioridade for suspensa ou encerrar, o sistema operacional permitirá que a tarefa interrompida retome sua execução a partir do ponto em que foi interrompida.

Sincronização de Tarefas e Eventos

O sistema operacional suporta três poderosos mecanismos para a sincronização de eventos. O gerenciador de semáforos fornece semáforos contadores com capacidade de enfileiramento e timeout. O gerenciador de eventos pode sincronizar vários eventos com mais de uma tarefa, oferecendo a melhor solução para a coordenação de eventos complexos. O sistema operacional oferece uma outra forma simples de sincronização de tarefas/eventos.

Uma tarefa pode aguardar incondicionalmente por um evento, chamando *Wait*. Este evento pode ser dependente de uma tarefa ou de tempo. A tarefa que está aguardando pelo evento permanece suspensa incondicionalmente até que outra tarefa execute uma chamada a *WakeTask*, requisitando ao sistema operacional que acorde a tarefa suspensa. O par *Wait*/*WakeTask* é usado para sincronizações de eventos simples.

O sistema operacional também suporta sincronização com timeout automático. Uma tarefa pode chamar *WaitTime*, especificando um intervalo de tempo máximo que deseja aguardar pela ocorrência de um evento. Se nenhuma outra tarefa executar *WakeTask* neste ínterim, o sistema operacional automaticamente acordará a tarefa ao expirar o intervalo. O sistema operacional passa para a tarefa uma indicação da ocorrência ou não do timeout.

Uma tarefa também pode ser sincronizada a uma outra tarefa através da passagem de mensagens. Uma tarefa envia uma mensagem para outra tarefa através de *SendMessageWait*, que força seu estado de espera. Quando a tarefa destino recebe a mensagem, pode executar *WakeCallingTask* a fim de acordar a tarefa que lhe enviou a mensagem. Esta tarefa, por sua vez, retomará sua execução sabendo que sua mensagem já foi recebida pela tarefa destino. Se a tarefa destino não executar *WakeCallingTask*, o sistema operacional acordará automaticamente a tarefa geradora da mensagem assim que a tarefa destino encerrar.

Supervisor de Interrupções

As tarefas executam com o sistema de interrupções do processador habilitado, o que permite ao sistema responder a eventos em tempo real.

Quando ocorre uma interrupção (como de um canal serial, por exemplo), o processador desabilita o sistema de interrupções e salva o endereço de retorno e seus flags correntes na pilha da tarefa interrompida, desviando para uma rotina de atendimento da interrupção.

Em geral, a rotina de atendimento da interrupção salva os registradores que necessita utilizar, atende a interrupção, restaura os registradores, habilita as interrupções e retorna ao programa que estava em execução no ponto em que o mesmo foi interrompido.

O supervisor de interrupções do sistema operacional simplifica a construção de rotinas de atendimento de interrupção...

Os seguintes serviços oferecidos pelo sistema operacional estão associados ao tratamento de interrupções:

Tratamento de interrupções	BeginInterrupt	Inicia o atendimento de uma interrupção
	ChangeInterruptHandler	Troca a rotina de atendimento de uma dada interrupção
	EndInterrupt	Termina o atendimento de uma interrupção
	GenerateInterrupt	Gera uma interrupção n de software n
	SetInterrupt	Instala um ponteiro de interrupção

Temporizações

Os sistemas de tempo real são caracterizados pela necessidade de fornecer um controle preciso sobre temporizações. O relógio de hardware do processador AL-2005 fornece a fonte básica de temporizações; o sistema operacional fornece o controle sobre seu uso.

A unidade de tempo do sistema operacional é o chamado tick do sistema, que é um intervalo de tempo fixo derivado do relógio de hardware.

> ATENÇÃO:
> O tick do sistema vale 10 ms.

Uma tarefa pode suspender a si própria por um intervalo de tempo específico (*WaitTime*). Pode também aguardar por um evento que deve ocorrer dentro de um intervalo de tempo determinado. Se o evento não ocorrer neste intervalo, a tarefa retoma sua execução com uma indicação de timeout.

Uma outra facilidade fornecida pelo sistema operacional, associada a controle de intervalos de tempo, são os temporizadores utilizados pelas aplicações. Uma vez criado, um temporizador pode ser disparado, verificado e parado por qualquer tarefa.

Os seguintes serviços oferecidos pelo sistema operacional estão associados a temporizadores:

Gerenciamento de temporizadores	CreateTimer	Cria um temporizador
	StartStopTimer	Dispara/para um temporizador
	TimeConvert	Converte milisegundos em ticks do sistema
	ReadTimer	Lê o valor corrente de um temporizador
	DeleteTimer	Remove um temporizador

Um temporizador deve ser “criado” por uma tarefa antes de poder ser utilizado (*CreateTimer*). O sistema operacional retorna um identificador do temporizador para a tarefa chamadora, que deve guardá-lo para futuras referências. Três parâmetros devem ser especificados na criação de um temporizador: seu período, um ponteiro para uma função de temporização, a ser executada quando esgotar o intervalo de tempo com o qual é inicializado no seu disparo, e um parâmetro opcional de 32 bits, dependente da aplicação.

O período do temporizador, especificado em ticks do sistema, determina se o mesmo é periódico ou não. Se o período for zero, o temporizador funciona uma única vez, cada vez que for disparado , permanecendo inativo até ser novamente disparado. Se o período for diferente de zero, o temporizador é periódico.

Sempre que esgota o intervalo de tempo programado em um temporizador, o sistema operacional executa a função de temporização associada ao mesmo na sua criação. Esta função recebe como parâmetros o identificador do temporizador e o parâmetro de 32 bits pré-definido na criação do temporizador.

Quando o temporizador é criado, o sistema operacional o coloca em um estado inativo, permanecendo neste estado até ser disparado por uma tarefa (*StartStopTimer*), que escreve seu valor inicial em ticks do sistema. Por conveniência, a função *TimeConvert* pode ser usada para converter um intervalo especificado em milisegundos para o número correspondente de ticks do sistema.

Os temporizadores são contadores decrementais. O sistema operacional decrementa contadores ativos (não zero) até que cheguem a zero. Quando um temporizador encerra sua contagem, o sistema chama a função de temporização associada. Temporizadores não periódicos permanecem inativos a menos que sejam novamente disparados pela função de temporização. Temporizadores periódicos são automaticamente disparados pelo sistema operacional com seus valores pré-definidos assim que a função de temporização encerrar sua execução.

Quando não houver mais necessidade de um temporizador, o mesmo pode ser removido (*DeleteTimer*).

Os seguintes serviços podem ser chamados pela função de temporização:

StartTask	Dispara a execução de uma tarefa
WakeTask	Acorda uma tarefa no estado de espera
SendMessage	Dispara uma tarefa pelo envio de uma mensagem
CreateTimer	Cria um temporizador
DeleteTimer	Remove um temporizador
ReadTimer	Lê o valor corrente de um temporizador
StartStopTimer	Dispara/para um temporizador
TimeConvert	Converte milisegundos em ticks do sistema
GetBuffer	Obtém um buffer de um pool específico
FreeBuffer	Libera um buffer
SignalEvent	Sinaliza um ou mais eventos em um grupo
SignalSemaphore	Sinaliza um semáforo
AddBottomList	Acrescenta ao final da lista
AddTopList	Acrescenta no início da lista
GetBottomList	Remove do final da lista
GetTopList	Remove do início da lista
ResetList	Inicializa uma lista circular

Mensagens

As mensagens são usadas para a comunicação entre tarefas cooperantes. Uma tarefa em execução pode requisitar a execução de uma outra tarefa mediante o envio de uma mensagem. Usualmente a ação executada pela tarefa destino variará de acordo com o conteúdo da mensagem.

O sistema operacional tem a capacidade de enfileirar requisições para execução de tarefas, permitindo com que mensagens associadas às requisições sejam empilhadas em caixas postais de uma forma controlada. Isto libera a tarefa requisitora a continuar com seu próprio processamento. Alternativamente, a tarefa requisitora pode suspender a si mesma até que a tarefa chamada tenha executado em resposta à requisição.

A tarefa requisitora pode enfileirar mensagens em até quatro níveis de prioridade. A tarefa destino pode portanto receber mensagens de um número qualquer de chamadores com as mensagens já ordenadas por prioridade pelo sistema operacional.

O projetista de uma aplicação descreve a necessidade exata de caixas postais de cada tarefa para o sistema operacional através da estrutura de descrição da tarefa. Para cada tarefa é possível especificar quais dos quatro níveis de prioridade são suportados, caso qualquer um deles for necessário. Além disto especifica-se também o número de mensagens possível em cada uma destas caixas postais.

O gerenciamento de caixas postais fornece o seguinte conjunto de serviços:

Gerenciamento de caixas postais	GetMailboxMessage SendMessage SendMessageWait	Obtém a mensagem de maior prioridade Inicia uma tarefa pelo envio de uma mensagem Inicia uma tarefa pelo envio de uma mensagem aguardando até que a tarefa receba a mensagem

Uma tarefa pode enviar uma mensagem para uma outra tarefa usando as funções *SendMessage* ou *SendMessageWait.*

Uma mensagem contém no máximo 12 bytes, totalmente dependentes da aplicação.

Exemplo:

```
char      c;
int   i;
int   iarray[6];
char      carray[12];
struct    msg {
    char    m1;
    char    m2;
    int     m3;
    long    m4;
} msga;
```

As seguintes chamadas enviarão as mensagens especificadas para a tarefa cujo identificador é *taskid*, com uma prioridade *priority*:

```
SendMessage (taskid, priority, &c);        /* Envia c        */
SendMessage (taskid, priority, &i);        /* Envia i        */
SendMessage (taskid, priority, iarray);    /* Envia iarray   */
SendMessage (taskid, priority, carray);    /* Envia carray   */
SendMessage (taskid, priority, &msga);     /* Envia msga     */
```

O chamador especifica a prioridade *priority* com a qual a mensagem deve ser enviada. O nível da prioridade (0 - maior, 1, 2 ou 3 - menor) determina a caixa postal na qual a tarefa chamada receberá a mensagem. O chamador identifica também a tarefa sendo chamada, especificando o seu identificador *taskid*.

O chamador pode suspender sua execução até que a tarefa chamada tenha sido executada em resposta à requisição usando *SendMessageWait*, ao invés de *SendMessage*.

O sistema operacional usa envelopes de mensagens para a passagem de mensagens. O sistema obtém um ponteiro para um envelope livre, move os parâmetros da mensagem para o envelope e insere o ponteiro na caixa postal da tarefa destino, no nível de prioridade especificado. Portanto, logo após o retorno da chamada de *SendMessage*, os parâmetros da mensagem já encontram-se livres para reutilização.

Ao receber a mensagem, a tarefa destino é disparada em seu endereço inicial pelo sistema operacional. A mensagem de 12 bytes é retirada do envelope, copiada para a pilha da tarefa e o envelope automaticamente liberado. Os parâmetros da mensagem recebida pela tarefa destino encontram-se exatamente na mesma ordem com a qual foram enviados.

Uma tarefa que recebe uma grande variedade de mensagens deve declarar uma união a fim de se referir a diferentes tipos de mensagens:

```
union     rmsg {
    char    c;
    int     i;
    int     iarray[6];
    char    carray[12];
    struct msg msga;
};
```

A tarefa pode ser escrita como:

```
void huge taskn (char c)                    /* declaração de parâmetro “fantasma” */
{
    union rmsg *pmsg;

    pmsg = (union rmsg *)&c;                /* ponteiro para a mensagem recebida */
    :
    :
    A mensagem recebida pode aqui ser acessada, usando o ponteiro pmsg.
    :
}
```

Neste exemplo, a tarefa que recebe mensagens não tem uma forma óbvia de determinar como a mensagem deve ser interpretada, isto é, se é apenas um caracter ou a estrutura *pmsg->msga* inteira. Esta questão normalmente é resolvida com a inclusão de um código de operação no início de cada mensagem.

Gerenciadores

O sistema operacional fornece um conjunto de gerenciadores para simplificar a sincronização de eventos, manipulação de recursos e alocação de memória.

Recursos

O gerenciador de **recursos** permite o acesso controlado a recursos definidos pelo usuário (como uma região de memória, por exemplo), isto é, torna possível o compartilhamento de recursos entre tarefas que executam concorrentemente. Utiliza um semáforo binário para limitar o acesso a cada recurso a uma única tarefa de cada vez. A posse e a liberação de um recurso é governada por chamadas ao gerenciador de recursos. O gerenciador de recursos oferece a característica única de identificar a tarefa proprietária de cada recurso, que é a única que tem permissão para liberá-lo.

O gerenciador de recursos difere do gerenciador de semáforos de duas formas. Não existe enfileiramento de prioridades. As tarefas esperam um recurso na base de que a primeira a requisitar é a primeira a ser atendida (‘first come, first served”). Não existe timeout. Uma tarefa aguardará incondicionalmente até que o recurso esteja disponível para seu uso privativo.

Além disto, diferentemente dos semáforos, a posse de um recurso está amarrada a uma tarefa específica. Apenas a tarefa que é dona de um recurso pode sinalizar sua liberação. Não é possível que mais de uma tarefa compartilhe a posse de um recurso em particular

O gerenciador de recursos oferece o seguinte conjunto de serviços:

Gerenciamento de recursos	CreateResource	Cria um recurso
	ReserveResource	Reserva um recurso
	ReleaseNestedResource	Libera um recurso (aninhado)
	ReleaseResource	Libera um recurso (incondicionalmente)
	DeleteResource	Remove um recurso

Um recurso deve ser criado por uma tarefa antes que possa ser utilizado (*CreateResource*). Cada recurso controlado pelo gerenciador de recursos tem um limite máximo, definido pelo usuário, do número de tarefas que podem aguardar por ele.

Uma tarefa requisita a posse de um recurso através da reserva do recurso (*ReserveResource*). Se o recurso estiver disponível, a tarefa tem assegurado imediatamente a posse do recurso. Se o recurso

não estiver disponível, a tarefa é colocada no final de uma lista de tarefas que estão aguardando por aquele recurso. Uma tarefa que já possui um recurso pode reservá-lo novamente, resultando em uma reserva aninhada.

Quando a tarefa que correntemente possui o recurso liberá-lo, o recurso será dado à tarefa que estiver há mais tempo aguardando por ele. A tarefa que possui o recurso deve liberá-lo uma vez para cada reserva aninhada que tenha realizado (*ReleaseNestedResource*) ou o recurso pode ser incondicionalmente liberado (*ReleaseResource*).

Se um recurso não necessita mais ser compartilhado, uma tarefa pode removê-lo (*DeleteResource*).

Semáforos

O gerenciador de **semáforos** fornece semáforos contadores de uso geral com enfileiramento de prioridades e timeout, que fornecem facilidades de sincronização entre tarefas executadas concorrentemente.

Em um ambiente multitarefa, freqüentemente é necessário que uma tarefa ganhe acesso mutuamente exclusivo a recursos críticos. O recurso pode ser uma região de código ou uma função não-reentrante (e portanto não compartilhável). Um semáforo pode ser usado para assegurar o acesso mutuamente exclusivo das tarefas a recursos críticos. Um semáforo também pode ser usado para controlar a alocação de recursos escassos.

E. W. Dijkstra introduziu duas operações primitivas para controlar o acesso exclusivo a recursos críticos. As primitivas abstratas, chamadas de operadores P e V, operam sobre uma variável chamada semáforo. Muitas variações destes operadores P e V foram implementadas deste sua introdução inicial. O gerenciador de semáforos fornece uma variação conhecida como semáforo contador ao qual é acrescentado o enfileiramento de prioridades e timeout automático.

O gerenciador de semáforos fornece o seguinte conjunto de serviços:

Gerenciamento de semáforos	CreateSemaphore	Cria um semáforo
	WaitSemaphore	Aguarda por um semáforo (timeout opcional)
	SignalSemaphore	Sinaliza um semáforo
	DeleteSemaphore	Remove um semáforo

Um número qualquer de semáforos definidos pelo usuário podem ser criados (*CreateSemaphore*). Ao ser criado, deve-se especificar sua contagem inicial e o número máximo de tarefas que podem aguardar por ele. Quando usado para exclusão mútua, o valor inicial do semáforo deve ser 1. Se o semáforo for inicializado com um valor *n*, pode ser usado para controlar o acesso a *n* recursos de um determinado tipo. Se o tamanho máximo da fila de espera pelo semáforo for igual a zero, não será permitido a nenhuma tarefa aguardar pelo semáforo se o recurso controlado por ele já estiver sendo utilizado.

Uma tarefa que deseja ganhar o controle de um semáforo ou utilizar um dos recursos gerenciados por ele requisita ao gerenciador de semáforos a concessão do mesmo (*WaitSemaphore*). Deve especificar a prioridade com que deseja aguardar, caso o semáforo não esteja imediatamente disponível. A tarefa será inserida em uma fila de espera pelo semáforo na prioridade especificada. Desta forma, tarefas que necessitam de maior prioridade no acesso a um recurso podem preemptar tarefas com prioridade de espera menor. A tarefa pode opcionalmente especificar o intervalo de tempo máximo, ou timeout, de espera, limitando desta forma o tempo que tem para aguardar pela liberação do semáforo para seu uso. Uma tarefa somente entrará em espera pelo semáforo caso a fila de espera do mesmo não se encontre cheia no momento da requisição.

Quando a tarefa encerra com sua utilização do recurso, sinaliza sua liberação (*SignalSemaphore*). O gerenciador de semáforos libera o recurso e verifica a fila de espera, sempre garantindo o acesso ao semáforo à tarefa que estiver no topo da fila. Se esta tarefa for de maior prioridade do que a tarefa que está liberando o recurso, ocorrerá um chaveamento de tarefas, dando à tarefa de maior prioridade

uma oportunidade imediata de utilizar o recurso. Caso a tarefa para a qual se concede o recurso for de menor prioridade do que a tarefa que o está liberando, a nova proprietária terá de aguardar até que a tarefa correntemente em execução renuncie ao controle do processador.

Se expirar o intervalo de timeout especificado pela tarefa, enquanto a mesma estiver aguardando por um semáforo, o gerenciador de semáforos remove-a da fila do semáforo e permite com que ela retome sua execução com uma indicação de timeout.

Quando não se necessitar mais de um semáforo, o mesmo pode ser removido (*DeleteSemaphore*).

Eventos

O gerenciador de **eventos** fornece um método conveniente para sincronizar uma ou mais tarefas a eventos detectados por outras tarefas. Uma tarefa pode requisitar ao gerenciador de eventos que suspenda sua operação até que um entre um conjunto de eventos ocorra. Alternativamente a tarefa pode pedir para esperar até que um conjunto completo de eventos ocorra. Opcionalmente a tarefa pode especificar um intervalo de timeout limitando o tempo que ficará aguardando pelo evento. Mais de uma tarefa podem estar aguardando por um evento ou conjunto de eventos.

Quando uma tarefa detecta a ocorrência de um evento, sinaliza o evento através de uma chamada ao gerenciador de eventos. O gerenciador de eventos verifica se o evento resultou em uma combinação de eventos para a qual uma ou mais tarefas estão aguardando. Se isto ocorrer, as tarefas que estiverem aguardando são liberadas para retomar sua execução.

O gerenciador de eventos fornece o seguinte conjunto de serviços:

Gerenciamento de eventos	GetEventGroup	Obtém o uso de um grupo de eventos
	WaitEvent	Aguarda evento(s) em um grupo (timeout opcional)
	SignalEvent	Sinaliza um ou mais eventos em um grupo
	FreeEventGroup	Libera um grupo de eventos

Cada grupo de eventos inclui 16 flags de evento. Cada evento de um grupo é representado por um flag booleano que constitui o estado do evento. Os 16 flags booleanos de evento são representados em uma palavra de 16 bits. Recomenda-se que os estados booleanos 0 e 1 sejam usados como indicadores de falso e verdadeiro, respectivamente. O estado zero representa portanto a ausência do evento. O estado um indica que o evento ocorreu.

Antes de poder ser utilizado, um grupo de eventos deve ser adquirido por uma tarefa (*GetEventGroup*). O gerenciador de eventos aloca um grupo de eventos e retorna um identificador do grupo para a tarefa chamadora, que deve guardá-lo para futuras referências. Ao adquirir um grupo de eventos, a tarefa deve especificar o valor inicial que cada um dos 16 eventos de um grupo devem assumir. A atribuição de eventos específicos a cada um dos flags de um grupo de eventos fica a critério do projetista.

Uma vez adquirido, um grupo de eventos pode ser usado para sincronizar tarefas a qualquer um dos 16 eventos que representa. Uma tarefa pode aguardar por um evento (*WaitEvent*), opcionalmente especificando um intervalo de tempo máximo no qual permanecerá aguardando pelo evento. Se desejar aguardar por mais de um evento, todos estes eventos devem estar contidos em um mesmo grupo.

Ao aguardar por eventos, a tarefa deve especificar o identificador do grupo de eventos desejado, uma máscara de 16 bits que identifica os eventos de interesse e um valor de 16 bits indicando o estado de interesse para cada um dos eventos selecionados. Especifica também um entre dois critérios a serem usados na detecção de eventos. A tarefa pode aguardar que qualquer um dos eventos selecionados no grupo atinja o estado especificado ou, alternativamente, que todos os eventos selecionados sejam iguais aos respectivos estados especificados.

Quando uma tarefa gera uma mudança de estado em um evento, sinaliza o evento (*SignalEvent*), especificando o identificador do grupo que contém o evento desejado. Mais de um evento podem ser

sinalizados de uma única vez. Para tanto deve-se especificar uma máscara de 16 bits identificando os eventos particulares do grupo e um valor de 16 bits especificando o novo estado de cada um dos eventos selecionados.

Sempre que um evento é sinalizado, o gerenciador de eventos determina se alguma tarefa está aguardando por eventos daquele grupo. Em caso afirmativo, verifica se o novo estado dos 16 flags de evento está de acordo com o critério de detecção desejado pela tarefa. Se isto ocorrer, a tarefa é acordada pelo gerenciador de eventos. Se não houver nenhuma outra tarefa de maior prioridade em execução, esta tarefa retomará imediatamente sua execução, com uma indicação de que a combinação de eventos que estava aguardando ocorreu.

Se não houver mais necessidade de um grupo de eventos, o mesmo pode ser removido (*FreeEventGroup*).

Buffers

O gerenciador de **buffers** fornece acesso rápido e eficiente a múltiplos conjuntos (pools) de buffers, onde cada buffer representa um bloco de tamanho fixo de memória. Esta forma de gerenciamento de memória vem ao encontro de aplicações em tempo real nas quais a disponibilidade de áreas de memória deve ser previsível e nas quais não se pode tolerar fragmentação de memória.

O gerenciador de buffers fornece o seguinte conjunto de serviços:

Gerenciamento de buffers	AddBufferUseCount	Incrementa contador de uso de um buffer
	CreateBufferPool	Cria um pool de buffers
	DeleteBufferPool	Remove um pool de buffers
	FreeBuffer	Libera um buffer
	GetBuffer	Obtém um buffer de um pool específico
	GetBufferSize	Obtém o tamanho de um buffer
	ResetBufferPool	Inicializa um pool de buffers específico
	ResetBufferPools	Inicializa todos os pools de buffers pré-definidos

Um pool de buffers consiste de um número qualquer de buffers de tamanho uniforme. Qualquer tamanho par de buffer, maior do que dois bytes e menor do que o limite de 64K de um segmento de memória é permitido. Todos os buffers em um pool devem residir no mesmo segmento de memória.

Um pool de buffers deve ser criado por uma tarefa (*CreateBufferPool*) antes de poder ser utilizado. O gerenciador de buffers retorna um identificador do pool para a tarefa chamadora, que deve guardá-lo para futuras referências. Ao criar um pool de buffers deve-se especificar três parâmetros: o número de buffers no pool, o tamanho de cada buffer e um ponteiro para a área de memória RAM onde ficarão localizados todos os buffers do pool. Esta área de memória é subdividida pelo gerenciador de buffers no número de buffers requisitado.

Todos os buffers no pool são encadeados através de uma lista de buffers livres e cada buffer tem associado a si um contador de uso, inicializado com zero para mostrar que não está sendo utilizado. Estas duas informações são mantidas internamente ao gerenciador de buffers.

Depois que um pool foi criado, pode-se obter um buffer do pool (*GetBuffer*). O gerenciador de buffers retira um buffer da lista de buffers livres, incrementa o contador de uso associado ao buffer e retorna um ponteiro para o primeiro byte do buffer. Pode-se então armazenar e recuperar dados deste buffer conforme o desejado.

Quando não se necessita mais de um buffer, o mesmo pode ser liberado (*FreeBuffer*). O buffer a ser liberado é identificado pelo ponteiro para seu primeiro byte, o mesmo que foi recebido quando o buffer foi adquirido. O gerenciador de buffers decrementa o contador de uso e se a contagem chegar a zero, o buffer é novamente encadeado à lista de buffers livres do pool ao qual pertence.

Uma vez adquirido um buffer, é possível incrementar seu contador de utilização (*AddBufferUseCount*), permitindo a existência simultânea de mais de um dono compartilhando o uso

do buffer. Na primeira vez em que se obtém um buffer, seu contador é setado em um. Se o contador de uso é incrementado de um, o buffer terá de ser liberado duas vezes antes de se tornar realmente livre. Esta característica pode ser importante para alguns tipos de aplicações, mas exige alguns cuidados. Quando um buffer possui vários donos executando concorrentemente, o conteúdo do buffer não deve ser alterado a menos que cada dono possa apenas escrever em uma parte do buffer exclusivamente sua. As únicas operações permitidas são a leitura e a liberação do buffer.

É possível também obter o tamanho de um buffer (*GetBufferSize*). Como o buffer foi obtido de um pool de buffers de tamanho conhecido, este serviço normalmente é desnecessário. Entretanto em aplicações onde a tarefa que utiliza um buffer não é necessariamente a mesma que o criou, é conveniente poder determinar seu tamanho.

Se em algum ponto não houver mais necessidade de nenhum dos buffers de um pool, o mesmo pode ser removido (*DeleteBufferPool*). Quando isto ocorrer, toda a área de memória RAM alocada na criação do pool de buffers é liberada.

Memória

O gerenciador de **memória** controla a alocação dinâmica de memória para as tarefas no ambiente multitarefa. Os seguintes serviços de gerenciamento de memória são fornecidos:

Gerenciamento da memória	AddBlockUseCount	Incrementa contador de uso de um bloco de memória
	FillBlock	Preenche um bloco de memória com um padrão
	FreeBlock	Libera um bloco de memória
	GetBlock	Obtém um bloco de memória
	GetBlockSize	Obtém o tamanho de um bloco de memória
	GetBlockUsingHandle	Obtém bloco de memória usando um handle privado
	GetHandle	Cria um handle privado de memória

A seguinte nomenclatura é adotada para o gerenciador de memória:

- **bloco de memória:** Região da memória RAM alinhada à parágrafo. Um bloco pode ultrapassar 64 Kbytes de tamanho.
- **contador de uso de um bloco:** Inteiro associado a um bloco de memória (mas que não faz parte dele). É usado pelo gerenciador de memória para manter-se informado sobre o número de donos do bloco.
- **handle de memória** Identificador de 32 bits fornecido pelo gerenciador de memória para identificar um bloco de memória privativo de uma tarefa, mas que ainda assim encontra-se sob o controle do gerenciador de memória

Qualquer tarefa pode chamar o gerenciador de memória a fim de obter um bloco de memória de qualquer tamanho (*GetBlock*). Se o gerenciador de memória for capaz de encontrar um bloco de memória de tamanho suficientemente grande que atenda o requerido, fornecerá um ponteiro para este bloco e uma indicação do tamanho real do mesmo. O bloco devolvido pode ser um pouco maior do que o tamanho requisitado.

Se o gerenciador de memória não localizar um bloco de tamanho suficiente, retorna uma indicação de erro e o tamanho do maior bloco disponível naquele instante. No entanto, se a tarefa imediatamente requisitar um bloco com este tamanho, a requisição pode novamente falhar, já que, em um ambiente multitarefa, outras tarefas com maior prioridade podem ter tomado parte desta memória antes mesmo que a tarefa possa renovar sua requisição.

Quando o gerenciador de memória aloca um bloco para ser utilizado por uma tarefa, seta o contador de uso do bloco em um. O dono de um bloco pode incrementar o contador de uso do bloco (*AddBlockUseCount*). Neste caso, o bloco deverá ser liberado duas vezes antes de se tornar efetivamente livre.

O contador de uso do bloco é a chave da posse de blocos de memória. O gerenciador de memória é dono de todos os blocos livres. Uma ou mais tarefas podem ser donas de um bloco já alocado. Este conceito de posse de um bloco é semelhante ao apresentado na seção anterior sobre a posse de um buffer através do gerenciador de buffers.

Quando o uso de um bloco de memória não é mais necessário, seu dono pode liberá-lo (*FreeBlock*), passando o mesmo ponteiro recebido quando o bloco foi originalmente alocado. O gerenciador de memória decrementa o contador de uso e, caso este tenha chegado a zero, retorna-o ao conjunto de blocos livres. Caso haja blocos livres adjacentes ao bloco liberado, todos estes blocos são reunidos para formar um único bloco maior.

O gerenciador de memória pode fornecer o tamanho de um bloco de memória em particular (*GetBlockSize*), o que pode ser útil se uma tarefa recebe a posse de um buffer de uma outra tarefa. O novo dono do bloco pode então verificar se o tamanho do bloco atende as suas necessidades. Se uma tarefa corromper os conteúdos de qualquer parte da memória fora dos limites do seu bloco, os efeitos são imprevisíveis e potencialmente desastrosos.

Uma característica particular do gerenciador de memória permite que qualquer bloco de memória possa ser tratado como uma seção de memória privada de onde blocos menores podem ser dinamicamente alocados. Para tanto, a tarefa chama a função *GetHandle*, passando-lhe um ponteiro para uma área privada de memória, cujo acesso deve ser controlado pelo gerenciador de memória. Deve-se especificar o tamanho desta área.

O gerenciador de memória converte esta área em uma seção de memória para uso privado da tarefa, identificada por um handle de memória, que é um ponteiro de 32 bits retornado para a tarefa que o requisitou.

O handle de memória deve ser usado para alocar blocos menores da seção de memória privada, através da função *GetBlockUsingHandle*. Quando uma seção de memória privada é criada, qualquer tarefa que tenha o handle pode adquirir blocos da seção. Depende apenas da tarefa que criou a seção privada determinar a que tarefas será dado o handle.

Listas

O gerenciador de **listas** fornece facilidades para manipulação de listas circulares genéricas.

Uma lista circular é uma estrutura de dados usada por uma aplicação para manter uma lista ordenada de bytes, palavras ou palavras duplas. Cada elemento é armazenado em um slot da lista. Cada lista contém um número fixo de slots, definido pelo usuário.

Os serviços do gerenciador de listas são reentrantes, permitindo com que sejam compartilhados por tarefas executando concorrentemente.

O gerenciador de listas fornece os seguintes serviços para manipulação de listas:

Gerenciamento de listas	AddBottomList AddTopList GetBottomList GetTopList ResetList	Acrescenta ao final da lista. Acrescenta no início da lista. Remove do final da lista. Remove do início da lista. Inicializa uma lista circular.

Uma lista circular é criada por uma aplicação através de uma chamada à função *ResetList*, para a qual deve fornecer três parâmetros: o número de slots da lista, o tamanho de cada slot (1, 2 ou 4 bytes) e um ponteiro para a uma área de memória onde a lista será armazenada, que deve ser alinhada à palavra e ter um tamanho de ($n*s$)+8 bytes, onde n é o número de slots e s o tamanho do slot (1, 2 ou 4).

O gerenciador de listas cria a lista na área de memória designada e torna-a vazia. Pode-se então acrescentar e remover elementos do tamanho definido, no início e/ou no final da lista, utilizando as

outras funções do gerenciador de listas (*AddBottomList*, *AddTopList*, *GetBottomList*, *GetTopList*). Estas funções devolvem o estado da lista a cada chamada. Quando acrescentando elementos à lista, pode-se saber se a lista já está cheia ou se acabou de ficar cheia com a inserção do elemento. Quando removendo elementos, pode-se saber se não há nenhum elemento na lista ou se a lista acabou de ficar vazia com a retirada do elemento.

Listas podem ser criadas dinâmica ou estaticamente. O exemplo a seguir ilustra listas estáticas de bytes, palavras e palavras duplas. *NSLOT* é definido como o número de slots em cada lista.

```
typedef char SLOT1          /* slot de 1 byte */
typedef int SLOT2           /* slot de 2 bytes */
typedef long SLOT4          /* slot de 4 bytes */

#define NSLOT               64

struct {
    int     header[4];
    SLOT1  slots[NSLOT];
    } bytelist;             /* lista circular com NSLOTS bytes de 8 bits */

struct {
    int     header[4];
    SLOT2  slots[NSLOT];
    } wordlist;             /* lista circular com NSLOTS palavras de 16 bits */

struct {
    int     header[4];
    SLOT4  slots[NSLOT];
    } ptrlist;              /* lista circular com NSLOTS ponteiros de 32 bits */
```

Entrada/Saída

Comunicação Serial

Os serviços de canais seriais foram implementados dentro de um conceito de “drivers” de dispositivo, de modo a suportar novas UARTs ou outras placas de comunicação que venham a ser utilizadas nos dois slots de expansão da placa AL-2005. No momento encontra-se implementado um driver para a interface serial National NS16550AF das placas satélites do processador AL-2005, que tanto podem obedecer ao padrão RS-232C quanto RS-485, de forma transparente para o “driver”.

Os seguintes serviços relacionados à comunicação serial são oferecidos:

Tratamento da comunicação serial	SerialConfig	Inicializa canal serial
	RXByte	Recebe um byte do canal serial
	TXByte	Transmite um byte pelo canal serial
	RXBlock	Recebe um bloco de bytes do canal serial
	TXBlock	Transmite um bloco de bytes pelo canal serial

Através destes serviços é possível configurar os canais seriais (*SerialConfig*), transmitir e receber bytes isolados (*TXByte*, *RXByte*) ou realizar comunicações blocadas (*TXBlock*, *RXBlock*).

O tamanho do buffer de recepção é de 128 bytes para versões até a 2.34, e de 512 bytes para as versões 2.35 ou superiores. Assim que são retirados dados do buffer, novos dados recebidos pela serial podem ser armazenados. Dados recebidos pela serial com o buffer lotado serão perdidos.

O tamanho do buffer de transmissão é de 1024 bytes. Assim que os dados (bytes) começam a ser inseridos no buffer é iniciada a sua transmissão. A cada byte transmitido é liberado um byte no buffer, deste modo pode-se transferir mais do que 1024 bytes sem riscos de perdas de dados.

Comunicando com o CP

O controle da comunicação com o CP fornece os seguintes serviços:

Tratamento da comunicação com o CP	HookPLC	Define a função de interface com chamada CHF do módulo F-2005 no programa aplicativo do CP
	GetIDCoproc	Obtém o número de identificação do processador
	GetNumTab	Obtém o número da tabela de configuração
	GetTamTab	Obtém o tamanho da tabela de configuração
	ReadOp	Lê operandos simples do CP
	ReadOpSwap	Lê operandos simples do CP
	WriteOp	Escreve em operandos simples do CP
	WriteOpSwap	Escreve em operandos simples do CP
	ReadTab	Lê tabela do CP
	ReadTabSwap	Lê tabela do CP
	WriteTab	Escreve em tabela do CP
	WriteTabSwap	Escreve em tabela do CP
	ReadTabs	Lê tabelas do CP
	WriteTabs	Escreve em tabelas do CP
	ReadPLC	Lê informações de status do CP ou da rede ALNET II
	WritePLC	Escreve informações de status no CP ou na rede ALNET II

A função *HookPLC* serve para definir uma rotina a ser usada como uma interface entre o programa aplicativo no processador AL-2005 e a instrução de chamada CHF do módulo F-2005.016, no programa aplicativo do CP. Esta rotina servirá para analisar as entradas da instrução, e eventualmente repassar a operação de reset ou configuração para outras tarefas. Caso isto não for necessário, a função *HookPLC* não precisa ser executada.

As funções *GetIDCoproc*, *GetNumTab* e *GetTamTab* obtém as seguintes informações armazenados pelo CP na área de memória compartilhada com o processador AL-2005: número de identificação do processador corrente, número e tamanho da tabela de configuração correntemente associada a um dado processador, respectivamente.

Uma tarefa pode ter acesso (leitura e escrita) a operandos e informações de status do CP através de chamadas às funções *ReadOp*, *ReadOpSwap*, *WriteOp*, *WriteOpSwap*, *ReadTab*, *ReadTabSwap*, *WriteTab*, *WriteTabSwap*, *ReadTabs*, *WriteTabs*, *ReadPLC* e *WritePLC*. Para tanto, deve alocar uma área de memória para transferir operandos ou informações de/para o CP. A tarefa pode opcionalmente aguardar pelo encerramento da transferência de dados, especificando um intervalo de tempo máximo no qual permanecerá aguardando.

Ao receber um pedido de comunicação com o CP, o sistema operacional enfileira a requisição, para tratá-la no momento em que o processador AL-2005 dispor da janela de tempo para acessar as áreas de comunicação com o CP.

As funções com Swap (*ReadOpSwap*, *WriteOpSwap*, *ReadTabSwap*, *WriteTabSwap*) são úteis para leitura de operandos do tipo memória (%M e %TM), tipo decimal (%D e %TD), tipo inteiro (%I e %TI) e tipo real (%F e %TF), pois fazem a inversão (swap) automático dos respectivos bytes que compõem as words (%M e %TM) ou double words (%D, %TD, %I, %TI, %F ou %TF), após ler ou

antes de escrever os dado no CP, visto que o CP trabalha no formato HI:LO e a BIOS no formato LO:HI.

Relógio e Sincronismo

A BIOS possui um relógio interno com resolução de 1ms, que pode ser acessado pela aplicação para realizar leituras ou escritas (acertos). A estrutura de uma variável do tipo relógio é a seguinte:

```
typedef struct
        {
         unsigned int  bios_mseg;     // milisegundo do tempo atual 0-999
         unsigned char bios_seg;      // segundo do tempo atual 0-59
         unsigned char bios_min;      // minuto do tempo atual 0-59
         unsigned char bios_hor;      // hora do tempo atual 0-23
         unsigned char bios_dia;      // dia da data atual 0-31
         unsigned char bios_mes;      // mes da data atual 1-12
         unsigned char bios_ano;      // ano da data atual 00-99
        } BIOSTIME;
```

Os seguintes serviços são oferecidos em relação ao relógio:

Gerenciamento do relógio da BIOS.	GetTime SetTime	Leitura do horário da BIOS. Escrita do horário na BIOS.

Além da manutenção do horário interno, a BIOS gera sinais de sincronismo através do canal serial de programação (RJ45), permitindo o acerto do relógio do CP através do cabo AL-2329. Os pulsos de sincronismos gerados a cada segundo tem curta duração (largura de 250ms), ou seja, impedem o escorregamento de horário entre a BIOS e o CP.

Os pulsos são gerados pelo sinal de RTS ("Request To Send") do canal serial de programação PG do AL-2005. O sinal gerado a cada segundo pode ser visto a seguir:

Outros Serviços

O sistema operacional fornece ainda os seguintes serviços de natureza geral que visam simplificar a programação de aplicações em tempo real no microprocessador 80186 da Intel:

Tratamento de ponto flutuante	EmuInit	Inicializa o tratamento de aritmética em ponto flutuante
Tratamento de funções de baixo nível	InByte InWord OutByte OutWord	Lê byte de uma porta de entrada Lê palavra de uma porta de entrada Escreve um byte em uma porta de saída Escreve uma palavra em uma porta de saída
	Offset Segment SetOffset SetSegment	Obtém o offset de um ponteiro Obtém o segmento de um ponteiro Seta o offset de um ponteiro Seta o segmento de um ponteiro

	Disable	Desabilita as interrupções
	Enable	Habilita as interrupções

5. Programação

Neste capítulo é apresentado um guia de referência das funções da biblioteca de suporte ao processador AL-2005. As principais funções disponíveis na API do AL-2005, já referenciadas no capítulo 4 - Sistema Operacional, correspondem à:

- funções para confecção de aplicativos multitarefa: a BIOS do AL-2005 é um sistema operacional multitarefa que fornece uma série de serviços aos aplicativos - criação de tarefas, espera de eventos, espera de recursos, semáforos, identificação de tarefas, etc
- funções de comunicação com acesso aos operandos do CP: permitem o acesso (escrita e leitura) aos operandos do CP (operandos simples e tabelas)
- funções de comunicação com acesso aos estados do CP: permitem o acesso (escrita e leitura) às informações de estado do CP e estado da sua rede de comunicação ALNET II
- funções de alocação de memória dinâmica
- funções de acesso a placa serial padrão RS-485 ou RS-232C

Estas funções são apresentadas em ordem alfabética e para cada uma delas são especificados:

- descrição
- sintaxe
- resultado
- observações (opcional)
- veja também

ATENÇÃO:
Para as funções desta lista, exceto quando explicitamente indicado, as interrupções são desabilitadas na entrada da função e posteriormente restauradas, ao encerrar a execução da função, para o estado em que se encontravam no momento da chamada.

Funções da Biblioteca de Suporte

AddBlockUseCount

Descrição
Incrementa contador de uso de um bloco de memória

Sintaxe
*STATUS huge AddBlockUseCount (char far *lpBlock, int nIncrement);*
onde:
lpBlock
ponteiro para um bloco de memória alocado pelas funções *GetBlock* ou *GetBlockUsingHandle*
nIncrement
valor com sinal a ser somado ao contador de uso do bloco de memória

Resultado
A função retorna:
OK
INVALID_MEMORY_BLOCK
MEMORY_BLOCK_NOT_IN_USE
MEMORY_BLOCK_USE_COUNT_OVERFLOW

Observações
- Se o retorno for diferente de *OK*, o contador de uso do bloco de memória permanece inalterado
- Se o contador de uso de um bloco for incrementado de *n*, o bloco não será liberado até que *n* chamadas a *FreeBlock* sejam executadas

Veja também
GetBlock, GetBlockUsingHandle, FreeBlock

AddBottomList

Descrição
Acrescenta ao final da lista

Sintaxe
*STATUS huge AddBottomList (void far *lpList, unsigned long lItem);*
onde:
lpList
ponteiro para uma lista circular (veja *ResetList*)
lItem
variável de 1, 2 ou 4 bytes a ser acrescentada à lista

Resultado
A função retorna:
0 = inserção ok; lista não está cheia
1 = inserção ok; lista agora ficou cheia
-*1* = não é possível inserir item; lista está cheia

Veja também
ResetList, AddTopList, GetTopList, GetBottomList

AddBufferUseCount

Descrição Incrementa contador de uso de um buffer

Sintaxe *STATUS huge AddBufferUseCount (char far *lpBuffer, int nIncrement);*
onde:
lpBuffer
ponteiro para um buffer obtido pela chamada da função *GetBuffer*
nIncrement
valor com sinal a ser somado ao contador de uso do buffer

Resultado A função retorna:
OK
BUFFER_NOT_IN_USE
BUFFER_USE_COUNT_OVERFLOW (contador+*nIncrement* > 65535)

Observações

- Se o contador de uso de um buffer for incrementado de *n*, o buffer não será retornado a lista de buffers livres do seu pool até que *n* chamadas a *FreeBuffer* sejam executadas.

Veja também *GetBuffer, FreeBuffer*

AddTopList

Descrição Acrescenta no início da lista

Sintaxe *STATUS huge AddTopList (void far *lpList, unsigned long lnItem);*
onde:
lpList
ponteiro para uma lista circular (veja *ResetList*)
lnItem
variável de 1, 2 ou 4 bytes a ser acrescentada à lista

Resultado A função retorna:
0 = inserção ok; lista não está cheia
1 = inserção ok; lista agora ficou cheia
-1 = não é possível inserir item; lista está cheia

Veja também *ResetList, AddBottomList, GetTopList, GetBottomList*

BeginInterrupt

Descrição Inicia o atendimento de uma interrupção

Sintaxe *void huge BeginInterrupt (void);*

Resultado O supervisor de interrupções salva todos os registradores na pilha do chamador. Se uma tarefa acabou de ser interrompida, o sistema operacional chaveia para sua pilha de interrupção antes de retornar para o chamador de *BeginInterrupt*

Veja também *EndInterrupt, SetInterrupt*

ChangeInterruptHandler

Descrição Troca a rotina de atendimento de uma dada interrupção

Sintaxe *void huge ChangeInterruptHandler (int nInterrupt, void interrupt (far *lpNewProc) (), void interrupt (far *(far *lplpOldProc)) ());*
onde:
nInterrupt
número da interrupção do microprocessador 80186 da Intel (0-255)
lpNewProc
ponteiro para a nova rotina de atendimento da interrupção
lplpOldProc
ponteiro para armazenar o ponteiro da rotina de atendimento anterior

Resultado Esta função NÃO DEVE ser utilizada para alterar a rotina de atendimento das interrupções de erro de divisão, “overflow” ou erro de limite

Veja também *SetInterrupt*

ChangeTaskPriority

Descrição Altera a prioridade de uma tarefa

Sintaxe *STATUS huge ChangeTaskPriority (TASKID TaskID, int nPriority);*
onde:
TaskID
identificador da tarefa cuja prioridade deve ser alterada
nPriority
prioridade desejada para a tarefa (1 = maior, até 127 = menor)

Resultado A função retorna:
OK
INVALID_TASK_ID
INVALID_TASK_PRIORITY (0 ou > 127)

Veja também *CreateTask, DeleteTask*

CreateBufferPool

Descrição Cria um pool de buffers

Sintaxe *STATUS huge CreateBufferPool (void far *pPoolDefinition, int far *lpnPoolID);*
onde:
pPoolDefinition
ponteiro para uma estrutura de definição do pool que descreve o pool de buffers a ser criado, como:

```
struct BufferPoolDefinition {
        char      *pPool;/* ponteiro p/ pool, alinhado à palavra */
        unsigned int nbuffers;   /* número de buffers no pool */
        unsigned int nsize;      /* tamanho de cada buffer em bytes */
};
```

A área alocada para o pool de buffers, além de estar alinhada à palavra, deve ter *N* bytes de memória RAM contínua, onde $N=((nbuffers*(nsize/2+2))+3)*2$ e $N<64$Kbytes.

lpnPoolID
ponteiro para armazenar o identificador do pool criado

Resultado A função retorna:
OK
NO_FREE_BUFFER_POOL
NO_BUFFERS_DEFINED
BUFFER_TOO_SMALL

Veja também *DeleteBufferPool*

CreateResource

Descrição Cria um recurso

Sintaxe *STATUS huge CreateResource (int QueuedTasks, int far *lpnResourceID);*
onde:
QueuedTasks
número máximo de tarefas que podem esperar pelo recurso em um dado instante (>=0 e <128)
lpnResourceID
ponteiro para armazenar o identificador do recurso criado

Resultado A função retorna:
OK
NO_FREE_RESOURCE_ID
INVALID_RESOURCE_QUEUE_DEPTH

Veja também *DeleteResource, ReserveResource, ReleaseResource*

CreateSemaphore

Descrição Cria um semáforo

Sintaxe

*STATUS huge CreateSemaphore (int far *lpSCB, int nValue, int nQueuedTasks);*
onde:
lpSCB
ponteiro para uma área de 4 palavras usada para controle do semáforo
nValue
valor inicial do semáforo (>=0 e < 128)
nQueuedTasks
número máximo de tarefas que podem esperar pelo semáforo em um dado instante (>=0 e <128)

Resultado

A função retorna:
OK
INVALID_SEMAPHORE_VALUE
INVALID_SEMAPHORE_QUEUE_DEPTH

Observações

- As interrupções não são alteradas.

Veja também

DeleteSemaphore, WaitSemaphore, SignalSemaphore

CreateTask

Descrição

Cria uma nova tarefa

Sintaxe

*STATUS huge CreateTask (TASKDESCR far *pDescription, TASKID *pTaskID);*
onde:
pDescription
ponteiro para o descritor da tarefa; todos os campos deste descritor devem ser válidos antes da chamada de *CreateTask*
pTaskID
ponteiro para armazenar o identificador da tarefa criada

Resultado

A função retorna:
OK
NO_FREE_TASK_CONTROL_BLOCK
INVALID_TASK_PRIORITY (0 ou >127)

Veja também

DeleteTask, KillTask, StopTask, ChangeTaskPriority, StartTask, SendMessage, SendMessageWait

CreateTimer

Descrição

Cria um temporizador

Sintaxe

*STATUS huge CreateTimer (unsigned int far *lpTimeID, unsigned int nPeriod, unsigned int huge (far *pProc) (unsigned int, long), long lnParam);*
onde:
lpTimeID
ponteiro para armazenar o identificador do temporizador criado
nPeriod

intervalo de tempo (em ticks de sistema) a ser usado se o temporizador for periódico; *nPeriod*=0, em caso contrário
pProc
ponteiro para a função de temporização a ser executada sempre que o temporizador esgotar o intervalo de tempo nele programado por *StartStopTimer*
lnParam
parâmetro de 4 bytes a ser passado para a função de temporização, sempre que a mesma for chamada pelo sistema operacional; usualmente é um ponteiro para alguma variável ou estrutura dependente da aplicação

Resultado A função retorna:
OK
NO_FREE_TIMER_BLOCK

Observações
- A criação de um temporizador não o dispara automaticamente. A função *StartStopTimer* deve ser usada para disparar o temporizador

Veja também *DeleteTimer, StartStopTimer, ReadTimer*

DeleteBufferPool

Descrição Remove um pool de buffers

Sintaxe *STATUS huge DeleteBufferPool (int nPoolID);*
onde:
nPoolID
identificador do pool a ser removido

Resultado A função retorna:
OK
INVALID_POOL_ID

Observações
- As interrupções não são alteradas
- Nenhuma operação que utilize os serviços do gerenciador de buffers e que venha a afetar os buffers do pool ou a alterar o conteúdo de qualquer um deles pode ocorrer durante ou após a remoção do pool

Veja também *CreateBufferPool*

DeleteResource

Descrição Remove um recurso

Sintaxe *STATUS huge DeleteResource (int nResourceID);*
onde:
nResourceID
identificador do recurso a ser removido

Resultado A função retorna:

OK
INVALID_RESOURCE_ID
RESOURCE_NOT_FREE

Veja também *CreateResource, ReserveResource, ReleaseResource*

DeleteSemaphore

Descrição Remove um semáforo

Sintaxe *STATUS huge DeleteSemaphore (int far *lpSCB);*
onde:
lpSCB
ponteiro para a área de controle do semáforo inicializada por *CreateSemaphore*

Resultado A função retorna:
OK
NO_SUCH_SEMAPHORE (*lpSCB* não foi inicializado por *CreateSemaphore*)
SEMAPHORE_IN_USE

Veja também *CreateSemaphore, WaitSemaphore, SignalSemaphore*

DeleteTask

Descrição Remove uma tarefa do sistema operacional

Sintaxe *STATUS huge DeleteTask (TASKID TaskID, int nPriority);*
onde:
TaskID
identificador da tarefa a ser removida
nPriority
prioridade de execução na qual a remoção deve ocorrer; a prioridade da remoção deve ser menor do que a prioridade de qualquer tarefa que possa afetar a tarefa sendo removida mas deve ser maior do que a prioridade de qualquer tarefa que fique permanentemente ativa (“compute bound”)

Resultado A função retorna:
OK
INVALID_TASK_ID
TASK_ABORT_NOT_ALLOWED

Observações

- Após a remoção, o identificador *TaskID* da tarefa não será mais válido
- Uma tarefa pode remover a si mesma
- Não se deve remover uma tarefa que esteja aguardando por um recurso, semáforo ou grupo de eventos

Veja também *ChangeTaskPriority, CreateTask*

DeleteTimer

Descrição Remove um temporizador

Sintaxe *void huge DeleteTimer (unsigned int TimerID);*
onde:
TimerID
identificador do temporizador a ser removido

Resultado Não há retorno de *DeleteTimer*.

Observações

- Uma função de temporização não deve chamar *DeleteTimer* para remover o temporizador ao qual está associada (ver *CreateTimer*). Pode, no entanto, remover quaisquer outros temporizadores aos quais não está associada
- Um temporizador ativo pode ser removido

Veja também *CreateTimer, StartStopTimer, RreadTimer*

Disable

Descrição Desabilita as interrupções

Sintaxe *void huge Disable (void);*

Resultado As interrupções são desabilitadas.

Observações

- As interrupções devem permanecer desabilitadas pelo menor período de tempo possível a fim de não degradar a performance do sistema.

Veja também *Enable*

EmuInit

Descrição Inicializa o tratamento de aritmética em ponto flutuante

Sintaxe *void far EmuInit (void);*

Resultado Não há retorno de *EmuInit*.

Observações

- Esta função deve ser chamada por cada tarefa que necessite fazer uso de valores em ponto flutuante, antes de utilizá-los

Enable

Descrição Habilita as interrupções

Sintaxe *void huge Enable (void);*

Resultado As interrupções são habilitadas.

Veja também *Disable*

EndInterrupt

Descrição Termina o atendimento de uma interrupção

Sintaxe *void huge EndInterrupt (void);*

Resultado O supervisor de interrupções verifica se a rotina de atendimento de interrupção está retornando para uma tarefa interrompida. Em caso positivo, chaveia para a pilha da tarefa. Os registradores salvos são restaurados da pilha antes de retornar para o chamador de *EndInterrupt*

Veja também *BeginInterrupt, SetInterrup*

EndTask

Descrição Termina a execução da tarefa corrente

Sintaxe *void huge EndTask (void);*

Resultado Não há retorno de *EndTask*

Observações

- Se alguma tarefa estiver aguardando pelo encerramento do processamento de uma mensagem enviada para esta tarefa, o sistema operacional automaticamente chama *WakeCallingTask* para acordá-la
- *EndTask* somente pode ser usada para terminar a tarefa que a chamar

Veja também *SendMessage, StartTask, WakeCallingTask, StopTask,KillTask, DeleteTask*

EscreveLeds

Descrição Permite controlar os LEDs PG, COM PG e ERR do painel do AL-2005.

Sintaxe *void huge EscreveLeds (char Valor, char Mascara);*
onde:
Valor
valor (tipo máscara) com o(s) LED(s) a ser(em) ligado(s):
PROG_LED
TXRX_LED
ERROR_LED
Mascara
Máscara com o(s) LED(s) afetado(s) pela escrita:
PROG_LED
TXRX_LED
ERROR_LED

Resultado Serão afetados somente os LEDs cuja máscara esteja habilitada. Para desligar um ou mais LEDs, basta passar 0 como parâmetro de valor, e como máscara o respectivo LED ou conjunto de LEDs

FillBlock

Descrição Preenche uma área de memória com um padrão

Sintaxe *STATUS huge FillBlock (MEMBLOCK pBlock, long lnSize, unsigned int nPattern);*
onde:
pBlock
ponteiro para a área de memória a ser preenchida com o padrão
lnSize
tamanho em bytes da área de memória a ser preenchida, apontada por *pBlock*
nPattern
padrão a ser usado no preenchimento

Resultado A função retorna:
OK
MEMORY_NOT_AVAILABLE

Observações

- As interrupções não são alteradas
- Esta função é particularmente útil para o preenchimento de grandes regiões de memória que excedam 64 Kbytes. A função é otimizada para velocidade no preenchimento de grandes regiões
- A função preencherá qualquer região de memória, alinhada a byte ou palavra, com um tamanho par ou ímpar. Os 8 bits menos significativos de *nPattern* são armazenados em **pBlock*. Os 8 bits mais significativos, em **(pBlock+1)*, desde que *lnSize* seja maior do que um. O padrão é então repetido até o tamanho indicado por *lnSize*

Veja também *GetBlock, FreeBlock*

FreeBlock

Descrição Libera um bloco de memória

Sintaxe *STATUS huge FreeBlock (char far *lpBlock);*
onde:
lpBlock
ponteiro para um bloco de memória alocado pelas funções *GetBlock* ou *GetBlockUsingHandle*

Resultado A função retorna:
OK
INVALID_MEMORY_BLOCK
MEMORY_BLOCK_NOT_IN_USE

Observações

- O chaveamento de tarefas é desabilitado na entrada da função e posteriormente restaurado ao encerrar a execução da função
- O contador de uso do bloco de memória é decrementado de um. Se o contador de uso chegar a zero, o bloco de memória é liberado para reutilização

Veja também *GetBlock, GetBlockUsingHandle, AddBlockUseCount*

FreeBuffer

Descrição Libera um buffer

Sintaxe *STATUS huge FreeBuffer (char far *lpBuffer);*
onde:
lpBuffer
ponteiro para um buffer obtido através da função *GetBuffer*

Resultado A função retorna:
OK
BUFFER_NOT_IN_USE

Observações

- O contador de uso do buffer é decrementado de um. Se o contador de uso chegar a zero, o buffer é retornado para a fila de buffers livres do pool

Veja também *GetBuffer, AddBufferUseCount*

FreeEventGroup

Descrição Libera um grupo de eventos

Sintaxe *STATUS huge FreeEventGroup (GROUP Group);*
onde:
Group
identificador do grupo de eventos obtido com uma chamada a *GetEventGroup*

Resultado A função retorna:
OK

EVENT_GROUP_IN_USE

Veja também *GetEventGroup, WaitEvent, SignalEvent*

GenerateInterrupt

Descrição Gera uma interrupção n de software n

Sintaxe *unsigned int huge GenerateInterrupt (int nInterrupt, BIOSREGS far *lpRegs);*
onde:
nInterrupt
tipo da interrupção (número 0 a 255)
lpRegs
estrutura contendo o valor dos registradores do processador necessários para a interrupção de software especificada, que se está gerando

Resultado A função retorna o valor do registrador *AX* no encerramento da rotina de atendimento da interrupção.

Veja também *BeginInterrupt,ChangeInterruptHandler, EndInterrupt, SetInterrupt*

GetBlock

Descrição Obtém um bloco de memória

Sintaxe *STATUS huge GetBlock (long lnSize, MEMBLOCK far *ppBlock, long far *lpSize);*
onde:
lnSize
tamanho em bytes de memória requerida
ppBlock
ponteiro para armazenar o ponteiro do bloco de memória retornado
lpSize
ponteiro para armazenar o tamanho útil real, em bytes, do bloco de memória em *ppBlock*; *lpSize* pode ser um pouco maior do que *lnSize*

Resultado A função retorna:
OK (*ppBlock* é um ponteiro para um bloco de memória; *lpSize* é o tamanho real do bloco)
MEMORY_NOT_AVAILABLE (*ppBlock* é indefinido; *lpSize* é o tamanho do maior bloco de memória correntemente disponível)

Observações
- O chaveamento de tarefas é desabilitado na entrada da função e posteriormente restaurado ao encerrar a execução da função
- Se a memória for alocada, o contador de uso do bloco é setado em um

Veja também *AddBlockUseCount, FreeBlock*

GetBlockSize

Descrição Obtém o tamanho de um bloco de memória

Sintaxe *STATUS huge GetBlockSize (MEMBLOCK pBlock, long far *lpSize);*
onde:
pBlock
ponteiro para um bloco de memória alocado pelas funções *GetBlock* ou *GetBlockUsingHandle*
lpSize
ponteiro para armazenar o tamanho efetivo, em bytes, do bloco de memória apontado por *pBlock*

Resultado A função retorna:
OK
INVALID_MEMORY_BLOCK
MEMORY_BLOCK_NOT_IN_USE

Veja também *GetBlock, GetBlockUsingHandle*

GetBlockUsingHandle

Descrição Obtém um bloco de memória usando um handle privado

Sintaxe *STATUS huge GetBlockUsingHandle (long lnSize, MEMBLOCK far *ppBlock, long far *lpSize, MEMHANDLE Handle);*
onde:
lnSize
tamanho em bytes de memória requerida
ppBlock
ponteiro para armazenar o ponteiro do bloco de memória retornado
lpSize
ponteiro para armazenar o tamanho útil real, em bytes, do bloco de memória em *ppBlock*; *lpSize* pode ser um pouco maior do que *lnSize*
Handle
handle de memória atribuído pelo gerenciador de memória em uma chamada prévia a *GetHandle* (o handle é um ponteiro)

Resultado A função retorna:
OK (*ppBlock* é um ponteiro para um bloco de memória; *lpSize* é o tamanho real do bloco)
MEMORY_NOT_AVAILABLE (*ppBlock* é indefinido; *lpSize* é o tamanho do maior bloco de memória correntemente disponível)
INVALID_MEMORY_BLOCK (*Handle* é um handle de memória inválido; *ppBlock* e *lpSize* são indefinidos)

Observações
- O chaveamento de tarefas é desabilitado na entrada da função e posteriormente restaurado ao encerrar a execução da função
- Se a memória for alocada, o contador de uso do bloco é setado em um

Veja também *GetHandle, AddBlockUseCount, FreeBlock*

GetBottomList

Descrição Remove do final da lista

Sintaxe *STATUS huge GetBottomList (void far *lpList, unsigned long far *lpItem);*
onde:
lpList
ponteiro para uma lista circular (veja *ResetList*)
lpItem
ponteiro para armazenar a variável de 1, 2 ou 4 bytes removida da lista; se *lpItem=NULL* (*0L*), o item é removido da lista mas não é retornado para a tarefa que chamou *GetBottomList*

Resultado A função retorna:
0 = remoção ok; lista não está vazia
1 = remoção ok; lista agora ficou vazia
-*1* = não é possível remover o item; lista está vazia

Veja também *ResetList, AddTopList, AddBottomList, GetTopList*

GetBuffer

Descrição Obtém um buffer de um pool de buffers específico

Sintaxe *STATUS huge GetBuffer (int nPoolId, char far *lpBuffer);*
onde:
nPoolId
identificador do pool de onde o buffer deve ser obtido
lpBuffer
ponteiro para armazenar o ponteiro do bufffer retornado

Resultado A função retorna:
OK
INVALID_POOL_ID
NO_BUFFER_AVAILABLE

Observações
- Se um buffer for alocado, o contador de uso do buffer é setado em um

Veja também *FreeBuffer, AddBufferUseCount, GetBufferSize*

GetBufferSize

Descrição Obtém o tamanho de um buffer

Sintaxe *unsigned int huge GetBufferSize (char far *lpBuffer);*
onde:
lpBuffer
ponteiro para um buffer obtido através da função *GetBuffer*

Resultado A função retorna o tamanho do buffer em bytes. Se o buffer não estiver sendo utilizado, é retornado 0

Observações
- As interrupções não são alteradas

Veja também *GetBuffer*

GetEventGroup

Descrição Obtém o uso de um grupo de eventos

Sintaxe *STATUS huge GetEventGroup (GROUP far *pGroup, unsigned int nInitialValue);*
onde:
pGroup
ponteiro para armazenar o identificador do grupo de eventos alocado
nInitialValue
valor inicial dos 16 flags de evento do grupo

Resultado A função retorna:
OK
NO_FREE_EVENT_GROUP

Veja também *FreeEventGroup, WaitEvent, SignalEvent*

GetHandle

Descrição Cria um handle privado de memória

Sintaxe *STATUS huge GetHandle (MEMBLOCK pBlock, long lnSize, MEMHANDLE far *pHandle);*
onde:
pBlock
ponteiro para uma área de memória a ser controlada pelo gerenciador de memória (pode-se usar blocos alocados através das funções *GetBlock* ou *GetBlockUsingHandle*)
lnSize
tamanho em bytes do bloco de memória apontado por *pBlock*
pHandle
ponteiro para armazenar um handle de memória a ser usado para acessos subsequentes a blocos menores dentro do bloco maior (o handle retornado é um ponteiro)

Resultado A função retorna:
OK
MEMORY_NOT_AVAILABLE (tamanho de memória muito pequeno ou excede 1 Mbytes)

Observações
- O chaveamento de tarefas é desabilitado na entrada da função e posteriormente restaurado ao encerrar a execução da função

Veja também *GetBlockUsingHandle, FreeBlock, AddBlockUseCount*

GetIDCoproc

Descrição Obtém o número de identificação do processador
Somente pode ser chamada de dentro da função HookPLC.

Sintaxe *char huge GetIDCoproc (void);*

Resultado A função retorna o número de identificação do processador no barramento (%Rxxxx / 8), podendo retonar valores de 0 a 7. Exemplo: %R000 = 0, %R0008 = .1, %R0016 = 2 etc.

Veja também *HookPLC, GetNumTab, GetTamTab*

GetMailboxMessage

Descrição Obtém a mensagem de maior prioridade

Sintaxe *STATUS huge GetMailboxMessage (int nPriority, char far *lpMessage);*
onde:
nPriority
prioridade da caixa postal na qual deve-se buscar a mensagem (0=maior; 3=menor)
lpMessage
nome de um array ou ponteiro para 12 bytes consecutivos para onde a mensagem será copiada caso esteja disponível; a área para cópia da mensagem pode ser qualquer estrutura de 12 bytes

Resultado A função retorna:
OK
CALLING_TASK_WAITING
NO_MESSAGE_WAITING

Observações
- Se o retorno for diferente de OK, a variável em lpMessage não é alterada
- Se o retorno for CALLING_TASK_WAITING, a tarefa que está chamando GetMailboxMessage está correntemente processando uma mensagem enviada por outra tarefa, que está por sua vez aguardando por uma confirmação de que a mensagem foi recebida. Não é possível obter nova mensagem antes que a mensagem corrente tenha sido completamente processada. Para tanto deve-se chamar WakeCallingTask para acordar a tarefa que está aguardando e só

então voltar a chamar GetMailboxMessage

Veja também *SendMessage, SendMessageWait, WakeCallingTask*

GetNumTab

Descrição Obtém o número da tabela de configuração associada a um processador
Somente pode ser chamada de dentro da função HookPLC.

Sintaxe *char huge GetNumTab (char num_coproc);*
onde:
num_coproc
número de identificação do processador

Resultado A função retorna o número da tabela de configuração associada ao processador especificado

Veja também *HookPLC, GetIDCoproc, GetTamTab*

GetTamTab

Descrição Obtém o tamanho da tabela de configuração associada a um processador
Somente pode ser chamada de dentro da função HookPLC.

Sintaxe *char huge GetTamTab (char num_coproc);*
onde:
num_coproc
número de identificação do processador

Resultado A função retorna o tamanho da tabela de configuração associada ao processador especificado

Veja também *HookPLC, GetIDCoproc, GetNumTab*

GetTaskDescriptor

Descrição Obtém o ponteiro para o descritor de uma tarefa

Sintaxe *STATUS huge GetTaskDescriptor (TASKID TaskID, TASKDESCR far * (far * lplpDescr));*
onde:
TaskID
identificador da tarefa que se deseja obter o ponteiro para o descritor
lplpDescr
ponteiro para armazenar o ponteiro para o descritor da tarefa

Resultado A função retorna:
OK

INVALID_TASK_ID

Observações
- As interrupções não são alteradas
- Se *TaskID* é inválido, é retornado o valor *NULL* (*0L*) em *lplpDescr*

Veja também *GetTaskID, LocateTask, DeleteTask*

GetTaskID

Descrição Obtém o identificador da tarefa corrente

Sintaxe *TASKID huge GetTaskID (void);*

Resultado A função retorna o identificador da tarefa correntemente em execução

Observações
- As interrupções não são alteradas

Veja também *GetTaskDescriptor, LocateTask*

GetTaskStatus

Descrição Obtém o estado de uma tarefa

Sintaxe *STATUS huge GetTaskStatus (TASKID TaskID, TASKSTATUS far *lpTaskStatus);*
onde:
TaskID
identificador da tarefa que se deseja obter o estado
lpTaskStatus
ponteiro para armazenar o estado da tarefa

Resultado A função retorna:
OK
INVALID_TASK_ID

GetTime

Descrição Leitura do relógio interno da BIOS.

Sintaxe *void huge GetTime (BIOSTIME far *lpTime);*
onde:
lpTime
ponteiro para a estrutura onde será armazenado o horário

Veja também *SetTime*

GetTopList

Descrição Remove do início da lista

Sintaxe *STATUS huge GetTopList (void far *lpList, unsigned long far *lpitem);*
onde:
lpList
ponteiro para uma lista circular (veja *ResetList*)
lpitem
ponteiro para armazenar a variável de 1, 2 ou 4 bytes removida da lista; se *lpItem=NULL* (*0L*), o item é removido da lista mas não é retornado para a tarefa que chamou *GetTopList*

Resultado A função retorna:
0 = remoção ok; lista não está vazia
1 = remoção ok; lista agora ficou vazia
-1 = não é possível remover o item; lista está vazia

Veja também *ResetList, AddTopList, AddBottomList, GetBottomList*

HookPLC

Descrição Define a função de interface com chamada CHF do módulo F-2005 no programa aplicativo do CP

Sintaxe *char huge HookPLC (char (*fptr_user)(int comando));*
onde:
fptr_user
ponteiro para a função a ser executada sempre que, no programa aplicativo do CP, ao se executar uma chamada CHF ao módulo F-2005 houver variação em alguma das entradas da instrução CHF
comando
parâmetro da função de tratamento das entradas da instrução CHF F-2005, apontada por *fptr_user*:
CHF_CONFIGURA (entrada **configura** ligada)
CHF_RESET (entrada **reset** ligada)
CHF_CONF_RES (entradas **configura** e **reset** ligadas)

Resultado A função retorna:
0 = *OK*
1 = esgotada a estrutura para armazenamento de ponteiros

Observações
- As funções de leitura e escrita de operandos, operandos tabela ou status não podem ser disparadas pelas funções de tratamento das entradas da instrução CHF

Veja também *GetIDCoproc, GetNumTab, GetTamTab*

InByte

Descrição Lê byte de uma porta de entrada

Sintaxe *char huge InByte (int nPort);*
onde:
nPort
número da porta de entrada

Resultado A função retorna o byte lido

Observações
- As interrupções não são alteradas

Veja também *InWord, OutByte, OutWord*

InWord

Descrição Lê palavra de uma porta de entrada

Sintaxe *unsigned int huge InWord (int nPort);*
onde:
nPort
número da porta de entrada

Resultado A função retorna a palavra lida

Observações
- As interrupções não são alteradas

Veja também *InByte, OutByte, OutWord*

KillTask

Descrição Mata uma tarefa
Força que uma tarefa pronta, em execução ou suspensa seja encerrada. Todas as mensagens nas caixas postais da tarefa serão descartadas.

Sintaxe *STATUS huge KillTask (TASKID TaskID);*
onde:
TaskID
identificador da tarefa a ser morta

Resultado A função retorna:
OK
INVALID_TASK_ID
TASK_ABORT_NOT_ALLOWED

Observações

- Qualquer tarefa que esteja aguardando que a tarefa morta responda a uma mensagem retomará sua execução
- Uma tarefa pode matar a si mesma
- Não se deve matar uma tarefa que esteja aguardando por um recurso, semáforo ou grupo de eventos

Veja também *EndTask, DeleteTask*

LocateTask

Descrição Localiza uma tarefa através de seu nome

Sintaxe *STATUS huge LocateTask (TASKID far *lpTaskID, char far *lpsTaskName);*
onde:
lpTaskID
ponteiro para armazenar o identificador da tarefa caso ela possa ser localizada
lpsTaskName
ponteiro para o tag de quatro caracteres que é o nome da tarefa que se deseja localizar

Resultado A função retorna:
OK
INVALID_TASK_ID

Observações

- Se mais de uma tarefa foram criadas com o mesmo tag, obtém-se o identificador de qualquer uma delas
- Todos os quatro caracteres em lpsTaskName devem ser iguais ao tag da tarefa para se obter um identificador válido

Veja também *GetTaskID, GetTaskDescriptor*

Offset

Descrição Obtém o offset de um ponteiro

Sintaxe *unsigned int huge Offset (void far *lptr);*
onde:
lptr
ponteiro para a memória

Resultado A função retorna um valor inteiro de 16 bits sem sinal representando a parte do offset de um ponteiro *lptr*

Observações

- As interrupções não são alteradas

Veja também *Segment, SetOffset, SetSegment*

OutByte

Descrição Escreve um byte em uma porta de saída

Sintaxe *void huge OutByte (int nPort, char Byte);*
onde:
nPort
número da porta de saída
Byte
byte a ser escrito

Observações
- As interrupções não são alteradas

Veja também *OutWord, InByte, InWord*

OutWord

Descrição Escreve uma palavra em uma porta de saída

Sintaxe *void huge OutWord (int nPort, int Word);*
onde:
nPort
número da porta de saída
Word
palavra a ser escrita

Observações
- As interrupções não são alteradas

Veja também *OutByte, InByte, InWord*

ReadOp

Descrição Lê operandos simples do CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge ReadOp (void *buffer, char tipo_op, int address, int sub, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória onde serão armazenados os valores do operando lido do CP
tipo_op
tipo do operando do CP:
TIPO_MEMORIA
TIPO_INTEIRO
TIPO_DECIMAL

TIPO_REAL
TIPO_E_S
TIPO_AUXILIAR

address
endereço do operando no CP

sub
endereço da subdivisão do operando do CP

num_op
número de operandos do CP que se deseja ler

nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela leitura dos operandos especificados; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da leitura, permanecendo a requisição na fila da BIOS para ser executada

Resultado

A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a leitura; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de leitura de um operando não declarado na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de leitura de um número inválido de operandos: 0 ou maior que o limite máximo permitido para o tipo)

Observação

- Com esta função é possível ler a seguinte quantidade de operandos:
  - 96 do tipo auxiliar (%A);
  - 96 do tipo entrada ou saída (%E/S);
  - 255 do tipo memória (%M);
  - 255 do tipo decimal (%D);
  - 255 do tipo inteiro (%I);
  - 255 do tipo real (%F).
- Se o tipo de operando for *TIPO_AUXILIAR* ou *TIPO_E_S*, o parâmetro *sub* deverá conter o número do bit (0 a 7) do octeto a ser lido por esta operação, ou o valor -1 para leitura do octeto inteiro. O valor lido é armazenado a partir do bit 0 do operando destino. O último byte de dado recebido deve ser mascarado pela aplicação, se seus bits não forem todos válidos.

Veja também

ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC

ReadOpSwap

Descrição Leitura de operandos simples tipo Memória, Decimal, Inteiro ou Real (%M, %D, %I ou %F) do CP, invertendo automaticamente os bytes de valores dos operandos para o formato utilizado pela BIOS
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge ReadOpSwap (void *buffer, char tipo_op, int address, int sub, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória onde serão armazenados os valores do operando lido do CP
tipo_op
tipo do operando do CP:
TIPO_MEMORIA
TIPO_INTEIRO
TIPO_DECIMAL
TIPO_REAL
Address
Endereço do operando no CP
Sub
Endereço da subdivisão do operando do CP
num_op
número de operandos do CP que se deseja ler
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela leitura dos operandos especificados; se *nTimeOut=0*, a tarefa retornará erro imediatamente, pois a BIOS não suporta o acúmulo de leituras com swap em sua fila de requisições, devendo-se aguardar sempre pela sua execução

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a leitura; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016; código de erro retornado quando *nTimeOut*=0)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de leitura de um operando não declarado na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de leitura de um número inválido de operandos: 0 ou maior que o limite máximo permitido para o tipo)

Observação
- Com esta função é possível ler até 4096 bytes de dados do CP, ou seja, até 2048 operandos memória (%M) ou 1024 operandos decimal (%D), inteiro (%I) ou real (%F).
- Os operandos com mais de um byte no CP (AL-2003 e AL-2004) estão no formato HI:LO, enquanto que no AL-2005 no formato LO:HI. Por isso da necessidade de conversão após a leitura e antes da escrita de operandos tipo

memória (%M e %TM), decimal (%D e %TD), inteiro (%I e %TI) e real (%F e %TF) no CP.

Veja também *ReadOp, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC*

ReadPLC

Descrição Lê informações de status do CP ou da rede ALNET II
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge ReadPLC (void *buffer, char tipo_req, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória onde serão armazenados as informações de status lidas do CP
tipo_req
tipo do operando do CP:
TIPO_STATUS_CP
TIPO_STATUS_ALNETII
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela leitura das informações especificadas; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da leitura

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a leitura; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)

Observações

- As estruturas de dados de armazenamento das informações de estado do CP e de estado da ALNET II devem ser do tipo *STATUS_PLC* e *STATUS_ALNETII* respectivamente:

```
typedef struct STATUS_PLC
{
  char  TipoUCP;              /* Tipo de UCP */
  char  Extensao[8];          /* Identificacao complementar */
  char  Versao[2];            /* Versao do executivo */
  char  ModoOperacao1;        /* Modo de operacao do CP */
  char  ModoOperacao2;        /* Modo de operacao complementar */
  char  CodigoMensagem[4];    /* Mensagens de erro ou advertencia */
  char  TempoCiclo[8];        /* Tempos de ciclo de execucao do CP */
```

```
  char  MaxTempoExec[1];     /* Tempos de execucao do ladder */
  char  IntE18E19[2];        /* Tempos de interrupcao E018 e E019 */
  char  Livre[2];
  char  EstadoRAM1;          /* Estado da memoria RAM */
  char  EstadoRAM2;          /* RAM compactada/nao compactada (0/1) */
  char  RAMLivre[16];        /* Memoria de RAM livre (memorias 0-7) */
  char  EstadoEPROM;         /* Estado da memoria EPROM */
  char  EPROMLivre[16];      /* Memoria de EPROM livre (memorias 0-7) */
  char  Reserva[100];
}

typedef struct STATUS_ALNETII
{
unsigned int NumeroTxOk;      /* Num de transmissoes sem erros */
unsigned int NumeroTxTcdt;    /* Num de TX com colisoes fora do preambulo */
unsigned int NumeroTxUdr;     /* Num de TX com underrun */
unsigned int NumeroTxNoack;   /* Num de TX sem ACK de hardware */
unsigned int NumeroTxErro;    /* Num de TX abortadas por excesso de erro */
unsigned int NumeroTxTimeout; /* Numero de time_out de buffers */
unsigned int NumeroTxFalta;   /* Numero de vezes que estourou buffer de TX */
unsigned int Livre0[2];

unsigned int NumeroRxOk;      /* Num de recepcoes sem erros */
unsigned int NumeroRxAbt;     /* Num de RX abortadas por colisao */
unsigned int NumeroRxOvr;     /* Num de RX com overrun */
unsigned int NumeroRxCrc;     /* Num de RX com erro de CRC */
unsigned int NumeroRxAe;      /* Num de RX com erro de alinhamento */
unsigned int NumeroRxLong;    /* Num de RX de pacotes muito grandes */
unsigned int NumeroRxTimeout; /* Num de time-outs em respostas de pacote */
unsigned int NumeroRxFalta;   /* Numero de vezes que estorou buffer de RX */
unsigned int Livre1[1];

unsigned int Velocidade;      /* Velocidade de comunicacao */
unsigned int EnderecoNo;      /* Endereco do CP na rede */
unsigned int EnderecoSubrede; /* Endereco da subrede local */
unsigned int Multicast;       /* Grupos de multicast que o CP pertence */
unsigned int Gateway1;        /* Endereco do primeiro gateway */
unsigned int Gateway2;        /* Endereco do segundo gateway */
unsigned int TimeoutLocal;    /* Time out local */
unsigned int TimeoutExterno;  /* Time out externo */
unsigned int TimeoutPacote;   /* Time out de pacotes */
unsigned int TentativasTx;    /* Numero de tentativas de retransmissao */
char         NomeEstacao[20]; /* Nome de identificacao do CP na rede */

unsigned int TipoConexao;     /* Tipo de conexao fisica (0=eletrica,1=otica) */
unsigned int Redundancia;     /* Redundancia da conexao (0=sem,1=com) */
unsigned int PeriodoTeste;    /* Periodo para envio de mensagem de teste */
                              /* para conexoes redundantes (em seg.) */
unsigned int TempoComutacao;  /* Tempo de espera para comutacao da conexao
(seg.) */
unsigned int Livre2[2];

unsigned int ConexaoSelec;    /* Conexao de comunicacao selecionada (1 ou 2)
*/
unsigned int EstadoConexao1;  /* Estado da conexao 1 (0=ok,1=falha) */
unsigned int EstadoConexao2;  /* Estado da conexao 2 (0=ok,1=falha) */
```

unsigned int Livre3[1]

unsigned int ConexaoForcada; / Conexao forcada (0=sem forcamento,1,2) */*
unsigned int EsdadoForcado1; / Estado forcado da conexao 1 (0=ok,1=falha) */*
unsigned int EsdadoForcado2; / Estado forcado da conexao 2 (0=ok,1=falha) */*
}

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, WritePLC*

ReadTab

Descrição Lê tabela do CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge ReadTab (void *buffer, char tipo_op, int address, int pos_ini, unsigned char num_op, unsigned char far *pnum_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória onde será armazenada a tabela lida do CP
tipo_op
tipo de tabela do CP:
- *TIPO_TABELA_MEMORIA*
- *TIPO_TABELA_INTEIRO*
- *TIPO_TABELA_DECIMAL*
- *TIPO_TABELA_REAL*

address
endereço da tabela do CP
pos_ini
posição inicial da tabela
num_op
número de operandos da tabela a serem lidos; se *num_op*=-1 (que para o tipo *unsigned char* equivale a 255) a tabela completa é lida e seu tamanho é devolvido no endereço apontado por *pnum_op*
pnum_op
ponteiro para armazenar o número total de operandos da tabela, se *num_op*=-1 (que para o tipo *unsigned char* equivale a 255)
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela leitura da tabela especificada; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da leitura, permanecendo a requisição na fila da BIOS para ser executada

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função

F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a leitura; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de leitura de uma tabela não declarada na memória do CP)
COD_ERRO_INDICE_TABELA_INVALIDO (houve a tentativa de leitura de uma posição de tabela não declarada na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de leitura de um número inválido de posições da tabela: 0)

Observações

- Com esta função é possível ler até 255 posições de tabela, seja ela do tipo memória (%TM), decimal (%TD), inteiro (%TI) ou real (%TF)
- A área de memória alocada para a leitura deve ser suficiente para acomodar toda a tabela especificada. Um erro fatal pode ocorrer caso sejam corrompidos os conteúdos de qualquer parte da memória fora dos limites da área alocada

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC*

ReadTabs

Descrição Lê tabelas do CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge ReadTabs (void *buffer, char tipo_op, int address, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória onde serão armazenadas as tabelas lidas do CP
tipo_op
tipo de tabela do CP:
TIPO_TABELA_MEMORIA
TIPO_TABELA_INTEIRO
TIPO_TABELA_DECIMAL
TIPO_TABELA_REAL
address
endereço da primeira tabela a ser lida do CP
num_op
número de tabelas a serem lidas
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela leitura das tabelas especificadas; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da leitura, permanecendo a requisição na fila da BIOS para ser executada

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é

provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a leitura; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de leitura de uma tabela não declarada na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de leitura de um número inválido de posições da tabela: 0 posições ou mais posições que o permitido por esta função)

Observações

- Com esta função é possível ler até 2048 bytes de dados do CP, ou seja, até 1024 posições de operandos tabela memória (%TM) ou até 512 posições de operandos tabela decimal (%TD), tabela inteiro (%TI) ou tabela real (%TF)
- A área de memória alocada para a leitura deve ser suficiente para acomodar todas as tabelas especificadas. Um erro fatal pode ocorrer caso sejam corrompidos os conteúdos de qualquer parte da memória fora dos limites da área alocada
- O número de posições de tabelas que serão lidas corresponde a soma de posições das N tabelas a serem lidas a partir da primeira tabela

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, WriteTabs, ReadPLC, WritePLC*

ReadTabSwap

Descrição Leitura de tabelas tipo Memória, Decimal, Inteiro ou Real (%TM, %TD, %TI ou %TF) do CP, invertendo automaticamente os bytes de valores dos operandos para o formato utilizado pela BIOS
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge ReadTabSwap (void *buffer, char tipo_op, int address, int pos_ini, unsigned char num_op, unsigned char far *pnum_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória onde será armazenada a tabela lida do CP
tipo_op
tipo de tabela do CP:
TIPO_TABELA_MEMORIA
TIPO_TABELA_INTEIRO
TIPO_TABELA_DECIMAL
TIPO_TABELA_REAL
address
endereço da tabela do CP
pos_ini
posição inicial da tabela
num_op
número de operandos da tabela a serem lidos; se *num*_op=-1, a tabela completa é

lida e seu tamanho é devolvido no endereço apontado por *pnum_op*
pnum_op
ponteiro para armazenar o número total de operandos da tabela, se *num*_op=-1
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela leitura da tabela especificada; se *nTimeOut=0,* a tarefa retornará erro imediatamente, pois a BIOS não suporta o acúmulo de leituras com swap em sua fila de requisições, devendo-se aguardar sempre pela sua execução

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a leitura; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016; código de erro retornado quando *nTimeOut*=0)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de leitura de uma tabela não declarada na memória do CP)
COD_ERRO_INDICE_TABELA_INVALIDO (houve a tentativa de leitura de uma posição de tabela não declarada na memória do CP ou quantidade de dados a transferir ultrapassa 2048 bytes)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de leitura de um número inválido de posições da tabela: 0)

Observações

- Com esta função é possível ler até 255 posições de tabela, seja ela do tipo memória (%TM), decimal (%TD), inteiro (%TI) ou real (%TF)
- A área de memória alocada para a leitura deve ser suficiente para acomodar toda a tabela especificada. Um erro fatal pode ocorrer caso sejam corrompidos os conteúdos de qualquer parte da memória fora dos limites da área alocada
- Os operandos com mais de um byte no CP (AL-2003 e AL-2004) estão no formato HI:LO, enquanto que no AL-2005 no formato LO:HI. Por isso da necessidade de conversão após a leitura e antes da escrita de operandos tipo memória (%M e %TM), decimal (%D e %TD), inteiro (%I e %TI) e real (%F e %TF) no CP

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC*

ReadTimer

Descrição Lê o valor corrente de um temporizador

Sintaxe *unsigned int huge ReadTimer (unsigned int TimerID);*

onde:
TimerID
identificador do temporizador a ser lido

Resultado A função retorna o valor do temporizador em ticks do sistema, isto é, o tempo restante antes do temporizador esgotar seu intervalo de tempo programado com *StartStopTimer*.

Observações

- As interrupções não são alteradas

Veja também *CreateTimer, DeleteTimer, StartStopTimer*

ReleaseNestedResource

Descrição Libera um recurso aninhado

Sintaxe *STATUS huge ReleaseNestedResource (int nResourceID);*
onde:
nResourceID
identificador do recurso a ser liberado

Resultado A função retorna:
OK
INVALID_RESOURCE_ID
RESOURCE_NOT_OWNED (o recurso não pode ser liberado pois não pertence à tarefa corrente)

Observações

- O contador de uso do recurso é decrementado de um a cada chamada a *ReleaseNestedResource*. O recurso não fica livre enquanto o contador de uso não chegar a zero
- Quando se tornar livre, o recurso será imediatamente repassado para a tarefa (se houver alguma) que estiver aguardando por ele há mais tempo. Se necessário, o reescalonamento de tarefas ocorre imediatamente

Veja também *CreateResource, DeleteResource, ReserveResource, ReleaseResource*

ReleaseResource

Descrição Libera um recurso incondicionalmente

Sintaxe *STATUS huge ReleaseResource (int nResourceID);*
onde:
nResourceID
identificador do recurso a ser liberado

Resultado A função retorna:
OK
INVALID_RESOURCE_ID
RESOURCE_NOT_OWNED (o recurso não pode ser liberado pois não pertence à

tarefa corrente)

Observações

- O contador de uso do recurso é zerado e o recurso liberado, podendo ser imediatamente repassado para a tarefa (se houver alguma) que estiver aguardando por ele há mais tempo. Se necessário, o reescalonamento de tarefas ocorre imediatamente

Veja também *CreateResource, DeleteResource, ReserveResource, ReleaseNestedResource*

ReserveResource

Descrição Reserva um recurso

Sintaxe

STATUS huge ReserveResource (int nResourceID);
onde:
nResourceID
identificador do recurso a ser reservado

Resultado

A função retorna:
OK
INVALID_RESOURCE_ID
RESOURCE_QUEUE_FULL (o recurso já é de outra tarefa e a fila de tarefas que estão aguardando por ele está cheia)

Observações

- Se o recurso já for de outra tarefa, a tarefa corrente será suspensa e ficará aguardando pelo recurso. Quando o recurso finalmente estiver disponível para a tarefa, a função retornará com *OK*
- O contador de uso do recurso é setado em um quando uma tarefa o recebe pela primeira vez. Uma tarefa que já possui um recurso pode reservá-lo novamente através de *ReserveResource*, resultando em uma reserva aninhada. O contador de uso do recurso é incrementado de um a cada nova reserva

Veja também *CreateResource, DeleteResource, ReleaseNestedResource, ReleaseResource*

ResetBufferPool

Descrição Inicializa (zera) um pool de buffers específico

Sintaxe

STATUS huge ResetBufferPool (int nPoolID);
onde:
nPoolID
identificador do pool a ser inicializado

Resultado

A função retorna:
OK
INVALID_POOL_ID
BUFFER_TOO_SMALL (o tamanho de buffer do pool é muito pequeno: é um erro fatal porque indica que os dados de controle do gerenciador de buffers estão corrompidos)

Observações
- As interrupções não são alteradas.
- Esta função apenas deve ser usada em aplicações nas quais é certo que os buffers não serão utilizados ou liberados durante a inicialização do pool
- Todos os buffers do pool são inicializados mesmo que estejam correntemente em uso. O conteúdo dos buffers será alterado

Veja também *ResetBufferPools*

ResetBufferPools

Descrição Inicializa (zera) todos os pools de buffers pré-definidos

Sintaxe *STATUS huge ResetBufferPools (void);*

Resultado A função retorna:
OK
BUFFER_TOO_SMALL (o tamanho de buffer de algum pool é muito pequeno: é um erro fatal porque indica que os dados de controle do gerenciador de buffers estão corrompidos)

Observações
- As interrupções não são alteradas
- Esta função apenas deve ser usada em aplicações nas quais é certo que os buffers não serão utilizados ou liberados durante a inicialização dos pools
- Todos os buffers dos pools são inicializados mesmo que estejam correntemente em uso. O conteúdo dos buffers será alterado

Veja também *ResetBufferPool*

ResetList

Descrição Inicializa uma lista circular

Sintaxe *void huge ResetList (void far *lpList, int SlotSize, int nSlots);*
onde:
lpList
ponteiro para a área de memória onde a lista circular será armazenada, como por exemplo, a seguinte estrutura estática:

```
typedef char SLOT1      /* slot de 1 byte */
#define NSLOT 64
struct {
        int       header[4];
        SLOT1  slots[NSLOT];
} bytelist;         /* lista com NSLOTS bytes de 8 bits */
```

SlotSize
tamanho de cada slot da lista (1, 2 ou 4 bytes)
nSlots
número de slots da lista (1 a 16380 slots)

Veja também *AddTopList, AddBottomList, GetTopList, GetBottomList*

ResetPendingWake

Descrição Reseta uma requisição pendente para acordar a tarefa corrente

Sintaxe *void huge ResetPendingWake (void);*

Resultado Esta função só afeta a tarefa corrente.

Observações
- As interrupções não são alteradas

Veja também *Wait, WaitTime, WakeTask, ResetTaskWake*

ResetTaskWake

Descrição Reseta uma requisição pendente para acordar uma tarefa

Sintaxe *STATUS huge ResetTaskWake (TASKID TaskID);*
onde:
TaskID
identificador da tarefa cuja requisição para ser acordada que se encontra pendente deve ser descartada

Resultado A função retorna:
OK
INVALID_TASK_ID

Observações
- As interrupções não são alteradas

Veja também *Wait, WaitTime, WakeTask, ResetPendingWake*

RXBlock

Descrição Recebe um bloco de bytes do canal serial

Sintaxe *STATUS huge RXBlock (char Channel, unsigned far *lpnBlock, unsigned far *lpnSize, TIME lnTimeOutRX);*
onde:
Channel
identificador do canal utilizado para receber o bloco:
COM_A ou *COM_B*
lpnBlock
ponteiro para a área da memória onde será armazenado o bloco recebido (a área

de memória deve ser capaz de armazenar 2**lpnSize* bytes)
lpnSize
ponteiro para o número de bytes a serem recebidos; no retorno, contém o número de bytes efetivamente transferidos para a área de memória da aplicação
lnTimeOutRX
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela recepção do bloco de byte; se *lnTimeOutRX=0*, trata-se de uma consulta à fila de recepção serial (ver **Resultado**)

Resultado

Cada valor armazenado na área apontada por *lpnBlock* ocupa 16 bits:

- os 8 bits mais significativos são bits de status; se um ou mais bits de status estiverem em 1, ocorreu um erro na recepção do byte correspondente; podem assumir os seguintes valores: *SERIAL_PARITY_ERROR*, *SERIAL_OVERRUN_ERROR* ou *SERIAL_FRAMING_ERROR*
- os 8 bits menos significativos contém o byte recebido, caso nenhum dos 8 bits mais significativos encontrar-se em 1

Se *lnTimeOutRX=0*, é realizada apenas uma consulta à fila de recepção serial:

- a área apontada por *lpnBlock* recebe, no máximo, o bloco de bytes requisitado, mas os bytes recebidos não são retirados da fila de recepção
- **lpnSize* recebe o número de bytes efetivamente devolvidos
- a função retorna imediatamente *SERIAL_OK*

A função retorna:
SERIAL_OK (recepção de bloco de bytes sem problemas)
SERIAL_INVALID_CHANNEL (erro de identificador de canal inválido)
SERIAL_UART_NOT_CONFIG (erro de canal não inicializado)
SERIAL_RX_TIME_OUT_ERROR (erro de timeout esgotado)
SERIAL_PARITY_ERROR (erro de paridade)
SERIAL_OVERRUN_ERROR (erro de “overrun”)
SERIAL_FRAMING_ERROR (erro de “framming”)
combinação (“OR”) de qualquer um dos três últimos valores

Observação

- O tamanho do buffer de recepcão é de 128 bytes para versões até a 2.34, e de 512 bytes para as versões 2.35 ou superiores. Dados recebidos pela serial com o buffer lotado serão perdidos

Veja também

SerialConfig, RXByte, TXByte, TXBlock

RXByte

Descrição

Recebe um byte do canal serial

Sintaxe

STATUS huge RXByte (char Channel, TIME lnTimeOutRX);
onde:
Channel
identificador do canal utilizado para receber o byte:
COM_A ou *COM_B*
lnTimeOutRX
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela recepção do byte; se *lnTimeOutRX=0*, trata-se de uma consulta à fila de recepção serial (ver **Resultado**)

Resultado

A função *RXByte* retorna um valor inteiro de 16 bits.

Os 8 bits mais significativos do valor de retorno são bits de status:
- se um ou mais bits de status estiverem em 1, ocorreu um erro
- se nenhum bit de status estiver em 1, o byte foi recebido sem erro

Os 8 bits menos significativos contém o byte recebido, caso nenhum dos 8 bits mais significativos encontrar-se em 1.

Se *lnTimeOutRX=0*, é realizada apenas uma consulta à fila de recepção serial:
- se não houver nenhum caracter na fila de recepção, é retornado imediatamente *SERIAL_RX_TIME_OUT_ERROR*
- se há caracter na fila de recepção, o caracter é retornado imediatamente, juntamente com *SERIAL_OK*, mas não é retirado da fila

A função retorna:
SERIAL_OK (recepção sem problemas; 8 bits menos significativos são o byte recebido)
SERIAL_INVALID_CHANNEL (erro de identificador de canal inválido)
SERIAL_UART_NOT_CONFIG (erro de canal não inicializado)
SERIAL_RX_TIME_OUT_ERROR (erro de timeout esgotado)
SERIAL_PARITY_ERROR (erro de paridade)
SERIAL_OVERRUN_ERROR (erro de “overrun”)
SERIAL_FRAMING_ERROR (erro de “framming”)

Observação

- O tamanho do buffer de recepcão é de 128 bytes para versões até a 2.34, e de 512 bytes para as versões 2.35 ou superiores. Dados recebidos pela serial com o buffer lotado serão perdidos

Veja também

SerialConfig, TXByte, RXBlock, TXBlock

Segment

Descrição

Obtém o segmento de um ponteiro

Sintaxe

*unsigned int huge Segment (void far *lptr);*
onde:
lptr
ponteiro para a memória

Resultado

A função retorna um valor inteiro de 16 bits sem sinal representando a parte do segmento de um ponteiro *lptr*

Observações

- As interrupções não são alteradas

Veja também

Offset, SetOffset, SetSegment

SendMessage

Descrição

Inicia uma tarefa pelo envio de uma mensagem

A tarefa requisita ao sistema operacional o envio de uma mensagem para outra tarefa, em uma dada prioridade, e que o mesmo inicie a execução da tarefa o mais cedo possível.

Sintaxe

*STATUS huge SendMessage (TASKID TaskID, int nPriority, char far *lpMessage);*
onde:
TaskID
identificador da tarefa para a qual a mensagem deve ser enviada
nPriority
prioridade da mensagem (0=maior; 3=menor)
lpMessage
nome de um array ou ponteiro para 12 bytes consecutivos que formam a mensagem a ser copiada para a tarefa destino; a mensagem pode ser qualquer estrutura de 12 bytes

Resultado

A função retorna:
OK
INVALID_TASK_ID
NO_FREE_ENVELOPE
NO_MAILBOX_DEFINED
MAILBOX_FULL

Observações

- Um chaveamento de tarefa imediato ocorrerá caso a tarefa, para a qual se está enviando a mensagem, for de maior prioridade do que a tarefa corrente

Veja também

SendMessageWait, StartTask, EndTask, WakeCallingTask

SendMessageWait

Descrição

Inicia uma tarefa pelo envio de uma mensagem aguardando até que a tarefa receba a mensagem

A tarefa requisita ao sistema operacional o envio de uma mensagem para outra tarefa, em uma dada prioridade, e que o mesmo inicie a execução da tarefa o mais cedo possível. A tarefa que faz a requisição é colocada em estado de espera até que:
1. a tarefa chamada recebe a mensagem e chama *WakeCallingTask* ou
2. a tarefa chamada recebe a mensagem e encerra sua execução chamando *EndTask* ou retornando para o sistema operacional

Sintaxe

*STATUS huge SendMessageWait (TASKID TaskID, int nPriority, char far *lpMessage);*
onde:
TaskID

identificador da tarefa para a qual a mensagem deve ser enviada
nPriority
prioridade da mensagem (0=maior; 3=menor)
lpMessage
nome de um array ou ponteiro para 12 bytes consecutivos que formam a mensagem a ser copiada para a tarefa destino; a mensagem pode ser qualquer estrutura de 12 bytes

Resultado

A função retorna:
OK
INVALID_TASK_ID
NO_FREE_ENVELOPE
NO_MAILBOX_DEFINED
MAILBOX_FULL

Observações

- Um chaveamento imediato ocorrerá para a tarefa pronta de maior prioridade

Veja também

SendMessage, StartTask, EndTask, WakeCallingTask

SerialConfig

Descrição

Inicializa canal serial

Sintaxe

STATUS huge SerialConfig (char Channel, unsigned nBaudRate, char nStopBits, char nParity, char nDataBits);
onde:
Channel
identificador do canal a ser inicializado:
    COM_A ou *COM_B*
nBaudRate
taxa de comunicação:
    SERIAL_BAUD50, SERIAL_BAUD75, SERIAL_BAUD110, SERIAL_BAUD134_5, SERIAL_BAUD150, SERIAL_BAUD300, SERIAL_BAUD600, SERIAL_BAUD1200, SERIAL_BAUD1800, SERIAL_BAUD2000, SERIAL_BAUD2400, SERIAL_BAUD3600, SERIAL_BAUD4800, SERIAL_BAUD7200, SERIAL_BAUD9600, SERIAL_BAUD19K, SERIAL_BAUD38K, SERIAL_BAUD56K, SERIAL_BAUD128K, SERIAL_BAUD256K, SERIAL_BAUD14400, SERIAL_BAUD28800, SERIAL_BAUD33600K
nStopBits
número de stop bits, controle de CTS/RTS/DCD e receiver FIFO trigger level:
    SERIAL_1STOPBIT (1 stop bit)
    SERIAL_2STOPBITS (2 stop bits)
    SERIAL_WAIT_CTS (aguarda o retorno de CTS na transmissão)
    SERIAL_RTS_ON (valor para o qual RTS deve ser setado ao encerrar a transmissão)
    SERIAL_MSR_INT (indica que devem ser geradas interrupcoes de Status Modem Register – DCD)
    SERIAL_TRIGGER_14 (14 bytes de FIFO de recepção, default)
    SERIAL_TRIGGER_08 (8 bytes de FIFO de recepção)
    SERIAL_TRIGGER_04 (4 bytes de FIFO de recepção)
    SERIAL_TRIGGER_01 (1 byte de FIFO de recepção)

nParity
tipo de paridade:
SERIAL_NOPARITY (sem bit de paridade)
SERIAL_ODDPARITY (paridade ímpar)
SERIAL_EVENPARITY (paridade par)
SERIAL_ALWAYS0 (bit de paridade sempre desligado)
SERIAL_ALWAYS1 (bit de paridade sempre ligado)
nDataBits
número de bits do dado:
SERIAL_5DATABITS, SERIAL_6DATABITS, SERIAL_7DATABITS, SERIAL_8DATABITS

Resultado

A função *SerialConfig* retorna um valor inteiro de 16 bits.

Os 8 bits mais significativos do valor de retorno são bits de status:
- se um ou mais bits de status estiverem em 1, ocorreu um erro
- se nenhum bit de status estiver em 1, não houve erro

Os 8 bits menos significativos identificam o erro ocorrido, caso algum dos 8 bits mais significativos encontrar-se em 1.

A função retorna:
SERIAL_OK (inicialização sem problemas; filas de recepção e transmissão resetadas)
SERIAL_INVALID_CHANNEL (identificador de canal inválido)
SERIAL_CFG_ERRO_BAUDRATE (taxa de comunicação inválida)
SERIAL_CFG_ERRO_STOPBITS (número de stop bits inválido)
SERIAL_CFG_ERRO_PARIDADE (tipo de paridade inválido)
SERIAL_CFG_ERRO_BITSDADOS (número de bits do dado inválido)
SERIAL_NO_HARDWARE (módulo AL-2405 não encontrado)

Observações

- O parâmetro *nStopBits* deverá conter o valor resultante do *OU lógico* entre o número de stop bits, sinais de modem e o modo de operação dos sinais de controle CTS e RTS

Veja também

RXByte, TXByte, RXBlock, TXBlock

SetInterrupt

Descrição

Instala um ponteiro de interrupção

Sintaxe

*void huge SetInterrupt (int nInterrupt, void interrupt (*) (), char far *lpArea);*
onde:
nInterrupt
número da interrupção do microprocessador 80186 da Intel (0-255)
interrupt
ponteiro para uma rotina de atendimento de interrupção
lpArea
ponteiro para uma área de memória estática com 16 palavras que não deve ser alterada enquanto o ponteiro de interrupção permanecer instalado

Resultado

Não há retorno de *DeleteTimer*.

Veja também *BeginInterrupt, EndInterrupt, ChangeInterruptHandler*

SetOffset

Descrição Seta o offset de um ponteiro

Sintaxe *void huge SetOffset (void far *lptr, int nOffset);*
onde:
lptr
ponteiro para a memória
nOffset
valor de offset a ser atribuído a *lptr*

Resultado A parte do offset do ponteiro *lptr* é setada com o valor de 16 bits sem sinal *nOffset*.

Observações
- As interrupções não são alteradas

Veja também *SetSegment, Offset, Segment*

SetSegment

Descrição Seta o segmento de um ponteiro

Sintaxe *void huge SetSegment (void far *lptr, int nSegment);*
onde:
lptr
ponteiro para a memória
nSegment
valor de segmento a ser atribuído a *lptr*

Resultado A parte do segmento do ponteiro *lptr* é setada com o valor de 16 bits sem sinal *nSegment*

Observações
- As interrupções não são alteradas

Veja também *SetOffset, Offset, Segment*

SetTime

Descrição Escrita no relógio interno da BIOS, para realizar acertos de horário.

Sintaxe *void huge SetTime (BIOSTIME far *lpTime);*
onde:
lpTime

ponteiro para a estrutura com o novo horário

Veja também *GetTime*

SignalEvent

Descrição Sinaliza um ou mais eventos em um grupo

Sintaxe *STATUS huge SignalEvent (GROUP Group, GROUP ValueMask, GROUP GroupValue);*
onde:
Group
identificador do grupo de eventos obtido com uma chamada a *GetEventGroup*
ValueMask
máscara de 16 bits identificando os flags de interesse no grupo de eventos
GroupValue
valor de 16 bits especificando o estado para cada um dos flags de evento selecionados pela máscara; o estado dos flags não selecionados pela máscara pode assumir qualquer valor

Resultado A função retorna:
OK
MAILBOX_FULL (não é possível sinalizar o(s) evento(s), provavelmente porque não existe envelopes de mensagem disponíveis)

Veja também *GetEventGroup, FreeEventGroup, WaitEvent*

SignalSemaphore

Descrição Sinaliza um semáforo

Sintaxe *STATUS huge SignalSemaphore (int far *lpSCB);*
onde:
lpSCB
ponteiro para a área de controle do semáforo inicializada por *CreateSemaphore*

Resultado A função retorna:
OK
NO_SUCH_SEMAPHORE (*lpSCB* não foi inicializado por *CreateSemaphore*)

Observações

- O semáforo será imediatamente repassado para a tarefa (se houver alguma) que estiver aguardando por ele no topo da sua fila de espera. Se necessário, o reescalonamento de tarefas ocorre imediatamente

Veja também *CreateSemaphore, DeleteSemaphore, WaitSemaphore*

StartStopTimer

Descrição Dispara ou para um temporizador

Sintaxe *void huge StartStopTimer (unsigned int TimerID, unsigned int TimerValue);*
onde:
TimerID
identificador do temporizador a ser disparado ou parado
TimerValue
intervalo de tempo em ticks do sistema a ser programado no temporizador; *TimerValue*=0 para o temporizador

Observações Uma função de temporização associada a um temporizador periódico pode pará-lo chamando *StartStopTimer* com *TimerValue*=0. O temporizador periódico pode ser novamente disparado chamando *StartStopTimer* com *TimerValue*=*n*. O temporizador esgotará após *n* ticks de sistema e retomará sua operação periódica com o seu período pré-definido por *CreateTimer*

Observações

- Temporizadores não periódicos (“one-shot timers”) podem ser re-disparados por sua rotina de temporização associada usando *StartStopTimer*

Veja também *CreateTimer, DeleteTimer, ReadTimer*

StartTask

Descrição Dispara a execução de uma tarefa

Sintaxe *STATUS huge StartTask (TASKID TaskID);*
onde:
TaskID
identificador da tarefa a ser disparada

Resultado A função retorna:
OK
INVALID_TASK_ID

Observações

- Um chaveamento imediato de tarefas ocorrerá caso a tarefa que está sendo disparada tiver maior prioridade do que a tarefa corrente

Veja também *SendMessage, EndTask*

StopTask

Descrição Para (termina) a execução de uma tarefa

Sintaxe
STATUS huge StopTask (TASKID TaskID);
onde:
TaskID
identificador da tarefa a ser parada

Resultado
A função retorna:
OK
INVALID_TASK_ID
TASK_ABORT_NOT_ALLOWED

Observações
- Qualquer tarefa que esteja aguardando que a tarefa parada responda a uma mensagem retomará sua execução
- Uma tarefa pode parar a si mesma
- Não se deve parar uma tarefa que esteja aguardando por um recurso, semáforo ou grupo de eventos

Veja também
KillTask, DeleteTask

TimeConvert

Descrição
Converte milisegundos em ticks do sistema

Sintaxe
unsigned int huge TimeConvert (unsigned int MiliSeconds);
onde:
MiliSeconds
número de milisegundos

Resultado
A função retorna o intervalo de tempo equivalente em ticks do sistema

Observações
- As interrupções não são alteradas
- Se a freqüência do relógio do sistema for tal que *MiliSeconds* requer mais de 65535 ticks de sistema, a função retorna 65535. Se *MiliSeconds* for menor do que metade de um tick de sistema, a função retorna 0

Veja também
CreateTimer, DeleteTimer, StartStopTimer, ReadTimer

TXBlock

Descrição
Transmite um bloco de bytes pelo canal serial

Sintaxe
*STATUS huge TXBlock (char Channel, char far *lpBlock, unsigned nSize, BOOL Wait);*
onde:
Channel
identificador do canal utilizado para transmitir o bloco:
COM_A ou *COM_B*
lpBlock
ponteiro para o bloco de bytes a transmitir
nSize

número de bytes a transmitir
Wait
indica se a função deve retornar imediatamente após a chamada (= *FALSE = 0*), aguardar o término da transmissão do bloco de bytes (= *TRUE = 1*) ou aguardar por um determinado tempo (>= *2*), especificado em ticks do sistema (funcionando como um tempo de timeout de transmissão)

Resultado A função retorna:
SERIAL_OK (transmissão do bloco de byte sem problemas)
SERIAL_TX_QUEUE_FULL (bloco de bytes foi enfileirado na fila de transmissão mas função teve de aguardar por posições livres)
SERIAL_INVALID_CHANNEL (identificador de canal inválido)
SERIAL_UART_NOT_CONFIG (canal não inicializado)
SERIAL_TX_TIME_OUT_ERROR (timeout de transmissão)

Observação
- O tamanho do buffer de transmissão é de 1024 bytes

Veja também *SerialConfig, RXByte, TXByte, RXBlock*

TXByte

Descrição Transmite um byte pelo canal serial

Sintaxe *STATUS huge TXByte (char Channel, char Byte, BOOL Wait);*
onde:
Channel
identificador do canal utilizado para transmitir o byte:
COM_A ou *COM_B*
Byte
byte a ser transmitido
Wait
indica se a função deve retornar imediatamente após a chamada (= *FALSE = 0*), aguardar o término da transmissão do byte (= *TRUE = 1*) ou aguardar por um determinado tempo (>= *2*), especificado em ticks do sistema (funcionando como um tempo de timeout de transmissão)

Resultado A função retorna:
SERIAL_OK (transmissão de byte sem problemas)
SERIAL_INVALID_CHANNEL (erro de identificador de canal inválido)
SERIAL_UART_NOT_CONFIG (erro de canal não inicializado)
SERIAL_TX_QUEUE_FULL (byte foi enfileirado na fila de transmissão mas função teve de aguardar por posição livre)
SERIAL_TX_TIME_OUT_ERROR (timeout de transmissão)

Observação
- O tamanho do buffer de transmissão é de 1024 bytes

Veja também *SerialConfig, RXByte, RXBlock, TXBlock*

Wait

Descrição Aguarda incondicionalmente até ser acordada

Sintaxe *void huge Wait (void);*

Resultado A tarefa será suspensa até que outra tarefa chame *WakeTask* a fim de acordar esta tarefa.

Observações

- As interrupções são habilitadas
- Se a tarefa tiver uma requisição pendente para acordar no momento em que chama *Wait*, continuará imediatamente sua execução sem espera
- Se houver alguma possibilidade de que alguma tarefa já fez uma chamada para *WakeTask* a fim de acordar a tarefa corrente, deve-se chamar *ResetPendingWake* para resetar qualquer requisição pendente antes de chamar *Wait*

Veja também *WaitTime, WakeTask, ResetPendingWake, ResetTaskWake, WaitSemaphore*

WaitEvent

Descrição Aguarda por evento(s) em um grupo

Sintaxe *STATUS huge WaitEvent (GROUP Group, GROUP ValueMask, GROUP GroupValue, GROUP MatchValue, unsigned int nTimeOut);*

onde:

Group

identificador do grupo de eventos obtido com uma chamada a *GetEventGroup*

ValueMask

máscara de 16 bits identificando os flags de interesse no grupo de eventos

GroupValue

valor de 16 bits especificando o estado para cada um dos flags de evento selecionados pela máscara; o estado dos flags não selecionados pela máscara pode assumir qualquer valor

MatchValue

critério de correspondência: se 0, qualquer flag selecionado que se encontrar no estado desejado é considerado como o evento de interesse; se <>0, todos os flags selecionados devem encontrar-se no estado especificado por *GroupValue* para se atingir o evento de interesse

nTimeOut

intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela ocorrência do evento especificado; se *nTimeOut=0*, a tarefa aguardará indefinidamente

Resultado As interrupções são habilitadas

A função retorna:

OK

TIME_OUT

Observações

- Se os eventos do grupo corresponderem ao critério especificado no momento da chamada de WaitEvent, a tarefa corrente continua a execução imediatamente sem esperar

Veja também *GetEventGroup, FreeEventGroup, SignalEvent*

WaitSemaphore

Descrição Aguarda por um semáforo

Sintaxe *STATUS huge WaitSemaphore (int far *lpSCB, unsigned int nInterval, int nPriority);*
onde:
lpSCB
ponteiro para a área de controle do semáforo inicializada por *CreateSemaphore*
nInterval
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela liberação do semáforo para seu uso; se *nInterval=0*, a tarefa aguardará indefinidamente
nPriority
prioridade com a qual a tarefa deseja aguardar pelo semáforo (0 é a maior e 255 a menor)

Resultado A função retorna:
OK
NO_SUCH_SEMAPHORE (*lpSCB* não foi inicializado por *CreateSemaphore*)
SEMAPHORE_WAIT_QUEUE_FULL
TIME_OUT

Observações

- O gerenciador de semáforos usa as funções *Wait* e *WakeTask* para aguardar por um semáforo. Se qualquer tarefa do sistema indiscriminadamente chamar *WakeTask* para acordar a tarefa que está aguardando pelo semáforo, esta tarefa retomará sua execução logo após a chamada de *WaitSemaphore*, com uma indicação de erro de *TIME_OUT*. Este erro será devolvido mesmo se *nInterval=0* (espera para sempre). O erro indica que a tarefa retomou sus execução sem ter garantida a posse do semáforo

Veja também *CreateSemaphore, DeleteSemaphore, SignalSemaphore*

WaitTime

Descrição Aguarda por um tempo determinado (“delay”)

Sintaxe *STATUS huge WaitTime (int nTicks);*
onde:
nTicks
intervalo de tempo de espera medido em ticks do sistema

Resultado As interrupções são habilitadas.

A função retorna:
OK
WAKENED_BEFORE_TIMEOUT (a tarefa foi acordada antes de encerrar o intervalo de tempo especificado)

Observações

- A tarefa pode retomar sua execução, sendo seu temporizador de espera parado, no momento que outra tarefa chamar *WakeTask* especificando esta tarefa
- Se a tarefa tiver uma requisição pendente para acordar no momento em que chama *WaitTime*, continuará imediatamente sua execução sem espera
- Se houver alguma possibilidade de que alguma tarefa já fez uma chamada para *WakeTask* a fim de acordar a tarefa corrente, deve-se chamar *ResetPendingWake* para resetar qualquer requisição pendente antes de chamar *WaitTime*

Veja também *Wait, WakeTask, ResetPendingWake, ResetTaskWake*

WakeCallingTask

Descrição Acorda uma tarefa que enviou uma mensagem para a tarefa corrente

Sintaxe *STATUS huge WakeCallingTask (void);*

Resultado A função retorna:
OK
INVALID_TASK_ID
CALLING_TASK_NOT_WAITING (a tarefa que enviou a mensagem não se encontra em estado de espera)
NO_MESSAGE_PROCESSING (a tarefa corrente não se encontra processando uma mensagem recebida de outra tarefa)

Observações

- Um chaveamento imediato de tarefas ocorrerá caso a tarefa que está sendo acordada tiver maior prioridade do que a tarefa corrente

Veja também *SendMessageWait, EndTask*

WakeTask

Descrição Acorda uma tarefa que se encontra no estado de espera (devido a uma chamada a *Wait* ou *WaitTime*)

Sintaxe *STATUS huge WakeTask (TASKID TaskID);*
onde:
TaskID
identifcador da tarefa a ser acordada

Resultado A função retorna:

OK
TASK_NOT_WAITING_WARNING (a tarefa que se deseja acordar não se encontra no estado de espera; a requisição para acordar fica pendente)
INVALID_TASK_ID
TASK_NOT_WAITING (a tarefa que se deseja acordar não se encontra no estado de espera e já há uma outra requisição para acordar pendente)

Observações

- Um chaveamento imediato de tarefas ocorrerá caso a tarefa que está sendo acordada tiver maior prioridade do que a tarefa corrente

Veja também *Wait, WaitTime, ResetPendingWake, ResetTaskWake, WaitSemaphore*

WriteOp

Descrição Escreve em operandos simples do CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge WriteOp (void *buffer, char tipo_op, int address, int sub, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área de memória que contém os valores a serem escritos no operando do CP
tipo_op
tipo do operando do CP:
TIPO_MEMORIA
TIPO_INTEIRO
TIPO_DECIMAL
TIPO_REAL
TIPO_E_S
TIPO_AUXILIAR
address
endereço do operando no CP
sub
endereço da subdivisão do operando do CP
num_op
número de operandos do CP que se deseja escrever
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela escrita dos operandos especificados; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da escrita, permanecendo a requisição na fila da BIOS para ser executada

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi

excedido antes da BIOS conseguir realizar a escrita; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de escrita de um operando não declarado na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de escrita de um número inválido de operandos: 0 ou maior que o limite máximo permitido para o tipo)

Observações

- Com esta função é possível escrever a seguinte quantidade de operandos:
  - 96 do tipo auxiliar (%A);
  - 96 do tipo entrada ou saída (%E/S);
  - 255 do tipo memória (%M);
  - 255 do tipo decimal (%D);
  - 255 do tipo inteiro (%I);
  - 255 do tipo real (%F).
- Se o bit 7 do parâmetro *tipo do operando do CP* estiver ligado e for *TIPO_MEMORIA*, *TIPO_DECIMAL*, *TIPO_INTEIRO ou TIPO_REAL*, a escrita será executada a nível de bit, servindo portanto, para escrita individual de bits de operandos memória, decimal, inteiro ou real
- Se o tipo de operando for *TIPO_AUXILIAR* ou *TIPO_E_S*, o parâmetro *sub* deverá conter o número do bit (0 a 7) do octeto a ser escrito por esta operação, ou o valor -1 para escrita do octeto inteiro
- Se o bit 7 do parâmetro de tipo do operando do CP estiver ligado (escrita orientada a bit) e o tipo de operando for *TIPO_MEMORIA*, o parâmetro *sub* deverá conter no byte alto o número do bit inicial (0 a 15) a partir do qual a primeira memória deverá ser escrita e no byte baixo o número do bit final (0 a 15) que deverá ser escrito na última memória escrita por esta operação
- Se o bit 7 do parâmetro de tipo do operando do CP estiver ligado (escrita orientada a bit) e o tipo de operando for *TIPO_DECIMAL*, *TIPO_INTEIRO ou TIPO_REAL*, o parâmetro *sub* deverá conter no byte alto o número do bit inicial (0 a 31) a partir do qual o primeiro operando deverá ser escrito e no byte baixo o número do bit final (0 a 31) que deverá ser escrito no último operando escrito por esta operação

Veja também *ReadOp, ReadOpSwap, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC*

WriteOpSwap

Descrição Escrita de operandos simples tipo Memória, Decimal, Inteiro ou Real (%M, %D, %I ou %F) no CP, invertendo automaticamente os bytes de valores dos operandos para o formato utilizado pelo CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge WriteOpSwap (void *buffer, char tipo_op, int address, int sub, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área de memória que contém os valores a serem escritos no operando do CP
tipo_op

tipo do operando do CP:

TIPO_MEMORIA
TIPO_INTEIRO
TIPO_DECIMAL
TIPO_REAL

address
endereço do operando no CP
sub
endereço da subdivisão do operando do CP
num_op
número de operandos do CP que se deseja escrever
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela escrita dos operandos especificados; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da escrita, permanecendo a requisição na fila da BIOS para ser executada

Resultado

A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a escrita; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de escrita de um operando não declarado na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de escrita de um número inválido de operandos: 0 ou maior que o limite máximo permitido para o tipo)

Observações

- Com esta função é possível escrever até 1024 bytes de dados no CP, ou seja, até 512 operandos memória (%M) ou 256 operandos decimal (%D), inteiro (%I) ou real (%F)
- Se o bit 7 do parâmetro *tipo do operando do CP* estiver ligado e for *TIPO_MEMORIA, TIPO_DECIMAL, TIPO_INTEIRO ou TIPO_REAL*, a escrita será executada a nível de bit, servindo portanto, para escrita individual de bits de operandos memória, decimal, inteiro ou real
- Se o bit 7 do parâmetro de tipo do operando do CP estiver ligado (escrita orientada a bit) e o tipo de operando for *TIPO_MEMORIA*, o parâmetro *sub* deverá conter no byte alto o número do bit inicial (0 a 15) a partir do qual a primeira memória deverá ser escrita e no byte baixo o número do bit final (0 a 15) que deverá ser escrito na última memória escrita por esta operação
- Se o bit 7 do parâmetro de tipo do operando do CP estiver ligado (escrita orientada a bit) e o tipo de operando for *TIPO_DECIMAL, TIPO_INTEIRO ou TIPO_REAL*, o parâmetro *sub* deverá conter no byte alto o número do bit inicial (0 a 31) a partir do qual o primeiro operando deverá ser escrito e no byte baixo o número do bit final (0 a 31) que deverá ser escrito no último operando escrito por esta operação

- Os operandos com mais de um byte no CP (AL-2003 e AL-2004) estão no formato HI:LO, enquanto que no AL-2005 no formato LO:HI. Por isso da necessidade de conversão após a leitura e antes da escrita de operandos tipo memória (%M e %TM), decimal (%D e %TD), inteiro (%I e %TI) e real (%F e %TF) no CP

Veja também *ReadOp, ReadOpSwap, WriteOp, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC*

WritePLC

Descrição Escreve informações de status no CP ou na rede ALNET II
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge WritePLC (void *buffer, char tipo_req, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória de onde serão lidas as informações de status a serem escritas no CP
tipo_req
tipo do operando do CP:
TIPO_STATUS_CP
TIPO_STATUS_ALNETII
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela escrita das informações especificadas; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da escrita

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a escrita; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)

Observações

- As estruturas de dados de armazenamento das informações de estado do CP e de estado da ALNET II devem ser do tipo *STATUS_PLC* e *STATUS_ALNETII* respectivamente (vide a função *ReadPLC* com a descrição das estruturas), mesmo que todas as suas variáveis não sejam utilizadas na escrita
- Para *TIPO_STATUS_CP* só é possível escrever no CP os tempos de ciclo de execução, variável *TempoCiclo*, composto pelo tempo de ciclo instatâneo, tempo de ciclo médio, tempo de ciclo máximo e tempo de ciclo mínimo
- Para *TIPO_STATUS_ALNETII* só é possível escrever no CP as seguintes variáveis da estrutura *STATUS_ALNETII*: *NumeroTxOk*, *NumeroTxTcdt*,

NumeroTxUdr, NumeroTxNoack, NumeroTxErro, NumeroTxTimeout, NumeroTxFalta, NumeroRxOk, NumeroRxAbt, NumeroRxOvr, NumeroRxCrc, NumeroRxAe, NumeroRxLong, NumeroRxTimeout, NumeroRxFalta, ConexaoForcada, EsdadoForcado1, EsdadoForcado2

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC*

WriteTab

Descrição Escreve em tabela do CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe *STATUS huge WriteTab (void *buffer, char tipo_op, int address, int pos_ini, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória que contém os valores a serem escritos na tabela do CP
tipo_op
tipo de tabela do CP:
TIPO_TABELA_MEMORIA
TIPO_TABELA_INTEIRO
TIPO_TABELA_DECIMAL
TIPO_TABELA_REAL
se o bit 7 deste parâmetro estiver ligado, a escrita será executada a nível de bit, servindo, portanto, para escrita individual de bits de tabela
address
endereço da tabela do CP
se for uma escrita orientada a bit (bit 7 do *tipo_op* estiver ligado), o byte alto deste parâmetro deverá indicar o número do último bit a ser escrito dentro da posição final de tabela (0 a 15)
pos_ini
posição inicial da tabela
se for uma escrita orientada a bit (bit 7 do *tipo_op* estiver ligado), o byte alto deste parâmetro deverá indicar o número do primeiro bit a ser escrito dentro da posição inicial de tabela informada no byte baixo deste parâmetro (0 a 15)
num_op
número de operandos da tabela a serem escritosse for uma escrita orientada a bit (bit 7 do *tipo_op* estiver ligado), o número de posições de tabela a serem escritas devem incluir tanto a posição inicial quanto a última posição que deve ser escrita, ainda que estas posições não tenham todos os seus bits escritos
se *pos_ini*=0 e *num_op*=-1 (que para o tipo *unsigned char* equivale a 255) a tabela completa é escrita, independente da quantidade de valores contidos no *buffer*
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela escrita da tabela especificada; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da escrita, permanecendo a requisição na fila da BIOS para ser executada

Resultado A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está

recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a escrita; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de escrita de uma tabela não declarado na memória do CP)
COD_ERRO_INDICE_TABELA_INVALIDO (houve a tentativa de escrita de uma posição de tabela não declarada na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de escrita de um número inválido de posições da tabela: 0)

Observações

- Com esta função é possível escrever até 255 posições de tabela, seja ela do tipo memória (%TM), decimal (%TD), inteiro (%TI) ou real (%TF)

Veja também

ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTabSwap, ReadTabs, WriteTabs, ReadPLC, WritePLC

WriteTabs

Descrição

Escreve em tabelas do CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe

*STATUS huge WriteTabs (void *buffer, char tipo_op, int address, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória que contém os valores a serem escritos na tabela do CP
tipo_op
tipo de tabela do CP:
TIPO_TABELA_MEMORIA
TIPO_TABELA_INTEIRO
TIPO_TABELA_DECIMAL
TIPO_TABELA_REAL
address
endereço da primeira tabela a ser escrita no CP
num_op
número de tabelas a serem escritas
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela escrita das tabelas especificadas; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da escrita, permanecendo a requisição na fila da BIOS para ser executada

Resultado

A função retorna:
OK (função executada com sucesso)

PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi excedido antes da BIOS conseguir realizar a escrita; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de escrita de uma tabela não declarado na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de escrita de um número inválido de posições da tabela: 0 posições ou mais posições que o permitido por esta função)

Observações

- Com esta função é possível escrever até 2048 bytes de dados do CP, ou seja, até 1024 posições de operandos tabela memória (%TM) ou até 512 posições de operandos tabela decimal (%TD), tabela inteiro (%TI) ou tabela real (%TF)
- O número de posições de tabelas que serão escritas corresponde a soma de posições das N tabelas a serem escritas a partir da primeira tabela

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, WriteTabSwap, ReadTabs, ReadPLC, WritePLC*

WriteTabSwap

Descrição Escrita de operandos tabela Memória, Decimal, Inteiro ou Real (%TM, %TD, %TI ou %TF) no CP, invertendo automaticamente os bytes de valores dos operandos para o formato utilizado pelo CP
Não pode ser chamada de dentro da função HookPLC.

Sintaxe

*STATUS huge WriteTabSwap (void *buffer, char tipo_op, int address, int pos_ini, unsigned char num_op, unsigned nTimeOut);*
onde:
buffer
ponteiro para a área da memória que contém os valores a serem escritos na tabela do CP
tipo_op
tipo de tabela do CP:
TIPO_TABELA_MEMORIA
TIPO_TABELA_INTEIRO
TIPO_TABELA_DECIMAL
TIPO_TABELA_REAL
se o bit 7 deste parâmetro estiver ligado, a escrita será executada a nível de bit, servindo, portanto, para escrita individual de bits de tabela.
address
endereço da tabela do CP
se for uma escrita orientada a bit (bit 7 do *tipo_op* estiver ligado), o byte alto deste parâmetro deverá indicar o número do último bit a ser escrito dentro da posição final de tabela (0 a 15)
pos_ini
posição inicial da tabela
se for uma escrita orientada a bit (bit 7 do *tipo_op* estiver ligado), o byte alto deste parâmetro deverá indicar o número do primeiro bit a ser escrito dentro da posição inicial de tabela informada no byte baixo deste parâmetro (0 a 15)
num_op
número de operandos da tabela a serem escritos
se for uma escrita orientada a bit (bit 7 do *tipo_op* estiver ligado), o número d posições de tabela a serem escritas devem incluir tanto a posição inicial quanto a última posição que deve ser escrita, ainda que estas posições não tenham todos os seus bits escritos.
se *pos_ini*=0 e *num_op*=-1 (que para o tipo *unsigned char* equivale a 255) a tabela completa é escrita, independente da quantidade de valores contidos no *buffer*
nTimeOut
intervalo de tempo máximo, em ticks de sistema, que a tarefa aguardará pela escrita da tabela especificada; se *nTimeOut=0*, a tarefa retornará imediatamente, sem aguardar o encerramento da escrita

Observação

- Com esta função é possível escrever até 255 posições de tabela, seja ela do tipo memória (%TM), decimal (%TD), inteiro (%TI) ou real (%TF).
- Os operandos com mais de um byte no CP (AL-2003 e AL-2004) estão no formato HI:LO, enquanto que no AL-2005 no formato LO:HI. Por isso da necessidade de conversão após a leitura e antes da escrita de operandos tipo memória (%M e %TM), decimal (%D e %TD), inteiro (%I e %TI) e real (%F e %TF) no CP.

Resultado

A função retorna:
OK (função executada com sucesso)
PLC_NOT_READY (comunicação inoperante com o CP: a BIOS não está recebendo as interrupções do CP, que permitem o acesso a sua memória; é provável que a função F-2005.016 não esteje sendo chamada pelo ladder)
WAITING_QUEUE_POSITION (a fila de requisições de comunicação de 16 posições da BIOS está lotada e esta comunicação não pôde ser atendida; verificar se não estão sendo feitas muitas leituras/escritas e poucas chamadas a função F-2005.016)
PLC_TIME_OUT (o tempo especificado no parâmetro *nTimeOut* da função foi

excedido antes da BIOS conseguir realizar a escrita; verificar com que freqüência estão sendo feitas chamadas a função F-2005.016)
COD_ERRO_TIPO_OPERANDO_INVALIDO (o tipo de operando especificado no parâmetro *tipo_op* da função não é um valor válido)
COD_ERRO_OPERANDO_NAO_DEFINIDO (houve a tentativa de escrita de uma tabela não declarado na memória do CP)
COD_ERRO_INDICE_TABELA_INVALIDO (houve a tentativa de escrita de uma posição de tabela não declarada na memória do CP)
COD_ERRO_NUMERO_OPERANDOS_INVALIDO (houve a tentativa de escrita de um número inválido de posições da tabela: 0)

Veja também *ReadOp, ReadOpSwap, WriteOp, WriteOpSwap, ReadTab, ReadTabSwap, WriteTab, ReadTabs, WriteTabs, ReadPLC, WritePLC*

6. Desenvolvendo uma Aplicação para o AL-2005

Instalação do Ambiente de Desenvolvimento

O ambiente de desenvolvimento de aplicações para o AL-2005 e duas aplicações exemplo são fornecidos juntamente com o processador AL-2005. As aplicações exemplo são modelos de desenvolvimento para aplicações que envolvam comunicação com outros dispositivos e de aplicações de cálculo efetuadas no coprocessador,

Os diretórios\arquivos necessários para o desenvolvimento de uma nova aplicação para o processador AL-2005 são instalados a partir do CD origem em um diretório de trabalho no microcomputador, gerando a árvore de diretórios descrita a seguir.

Diretório INCLUDE

Contém unicamente o arquivo AL2005.H de cabeçalho para aplicações do AL-2005.

\INCLUDE	diretório com o header do AL-2005
AL2005.H	arquivo de Header para programas AL-2005

Diretório LIB

Contém ambiente para a geração das bibliotecas de serviços especificos para o processador AL-2005.

\LIB	diretório c/ bibliotecas e startup para o AL-2005
AL2005L.LIB	biblioteca de serviços modelo Large
AL2005S.LIB	biblioteca de serviços modelo Small
C0DBGL.OBJ	código de startup para modelo Large - depuração
C0DBGS.OBJ	código de startup para modelo Small - depuração
C0EXEL.OBJ	código de startup para modelo Large - execução
C0EXES.OBJ	código de startup para modelo Small - execução

Diretório UTIL

Contém o utilitário LOCATE para o desenvolvimento das aplicações.

\UTIL	diretório de utilitários
LOCATE.EXE	executável para realocação de aplicações (utilizado em modo depuração)

Diretório AL3860

Contém o programa carregador de aplicativos.

\AL3860	diretório do carregador
AL3860.EXE	programa carregador de aplicativos
AL3860.INI	arquivo de inicialização

Diretório SUPORTE

Contém programa para teste das interfaces seriais A e B.

\SUPORTE	diretório de suporte
TSTCOM.TXT	instruções de utilização dos programas
TSTCOM_A.EXE	programa para teste da porta COM A
TSTCOM_B.EXE	programa para teste da porta COM B

Diretório TD2005

Para geração de uma nova versão, deve-se chamar o compilador Borland C++ (BC), o qual invocará a aplicação para depuração no AL-2005, a qual deve ser carregada via carregador AL-3860.

\TD2005	diretório com a aplicação de depuração
TD2005.EXE	aplicação depurador remoto para AL-2005

Diretório DOCS

Para geração de uma nova versão, deve-se chamar o compilador Borland C++ (BC), o qual invocará a aplicação para depuração no AL-2005, a qual deve ser carregada via carregador AL-3860.

\DOCS	diretório contendo documentação do produto
TUTORIAL.PDF	tutorial apresentando uma visão geral do produto
MU207006.PDF	manual de utilização do produto.

Diretórios DEMOCOM e DEMOCALC

O diretório DEMOCOM contém todo o ambiente de desenvolvimento de um programa exemplo que envolve comunicações a ser executado no AL-2005.

O diretório DEMOCALC contém o ambiente de desenvolvimento de um programa exemplo para cálculo.

\DEMOCOM	diretório com a aplicação exemplo para drivers de comunicação
\DEMOCALC	diretório com a aplicação exemplo para cálculo
\AL2003	diretório com o projeto demo para o AL-2003
C-DEMO.000	modulo de configuração
DEMO.MTL	projeto do MasterTool
E-DEMO.000	módulo de partida
E-DEMO.001	módulo de execução principal
F-2005.016	função de comunicação com o AL-2005
\AL2004	diretório com o projeto demo para o AL-2004
C-DEMO.000	modulo de configuração
DEMO.MTL	projeto do MasterTool
E-DEMO.000	módulo de partida
E-DEMO.001	módulo de execução principal
F-2005.016	função de comunicação com o AL-2005
DEMO.C	fonte de programa exemplo para AL-2005
PSDFLOAT.C	somente DEMOCALC – API para cálculo ponto flutuante
PSDFLOAT.H	somente DEMOCALC – API para cálculo ponto flutuante
FLOAT.ASM	somente DEMOCALC – API para cálculo ponto flutuante
DEMO.DSK	configuração do projeto no Borland C++
DEMO.EXE	aplicação executável no AL-2005
DEMO.H	arquivo de header para programa exemplo
DEMO.PRJ	arquivo de projeto Borland C++ para gerar executável
DEMOTD.DSK	configuração do projeto no Borland C++
DEMOTD.EXE	aplicação depurável no AL-2005
DEMOTD.PRJ	arquivo de projeto Borland C++ para gerar depurável
RELOCA.CFG	arquivo de configuração do Paradigm LOCATE
XDEMOTD.EXE	aplicação relocada depurável no AL-2005 em conjunto com o Turbo Debugger)

O projeto DEMO.MTL deve ser enviado para a UCP através do programador MasterTool.

Deve ser verificado qual o modelo de UCP utilizado e selecionado o projeto correspondente.

A figura 6-1 mostra a inicialização da tabela de configuração – %TM0000 - passada como parâmetro de entrada 1 na chamada do módulo F-2005.016. Esta tabela é referida também como tabela de relações quando efetua a correlação entre operandos de protocolos distintos. A decodificação das informações constantes nesta tabela é efetuada pelo software. A tabela de configuração das aplicações DEMO é apresentada como comentário do arquivo DEMO.H de cada aplicação .

A figura 6-2 mostra a primeira chamada do módulo F-2005.016, contida no módulo principal de execução.

Na sequência, as figuras 6-3 e 6-4 mostram duas opções de configuração dos parâmetros de entrada do módulo F-2005.016. Quando o parâmetro de entrada 2 - %KM é 00000, o AL-2005 estará executando em modo *release*, isto é, não permitirá a depuração remota da aplicação carregada. Quando %KM vale +00019, o processador estará apto a depuração remota da aplicação.

Para descrição completa dos parâmetros da CHF consulte o Capítulo 3, Configuração.

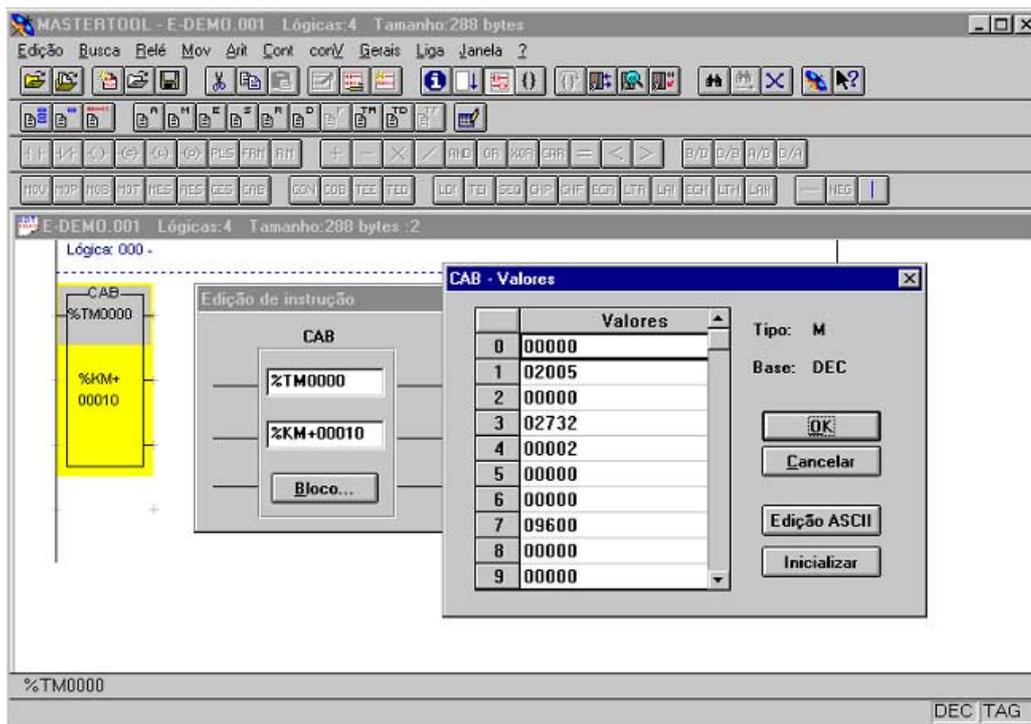



Figura 6-1 Inicialização da Tabela de Configuração da Aplicação

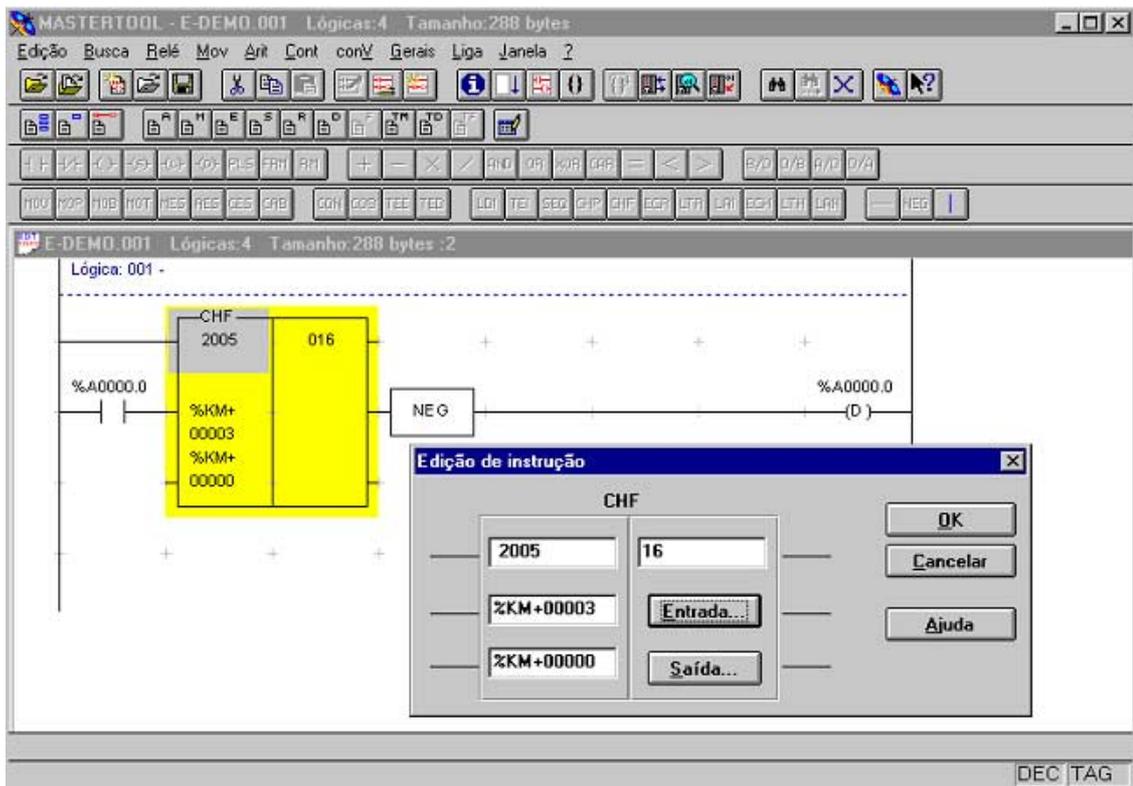



Figura 6-2 Chamada do Módulo F –2005.016

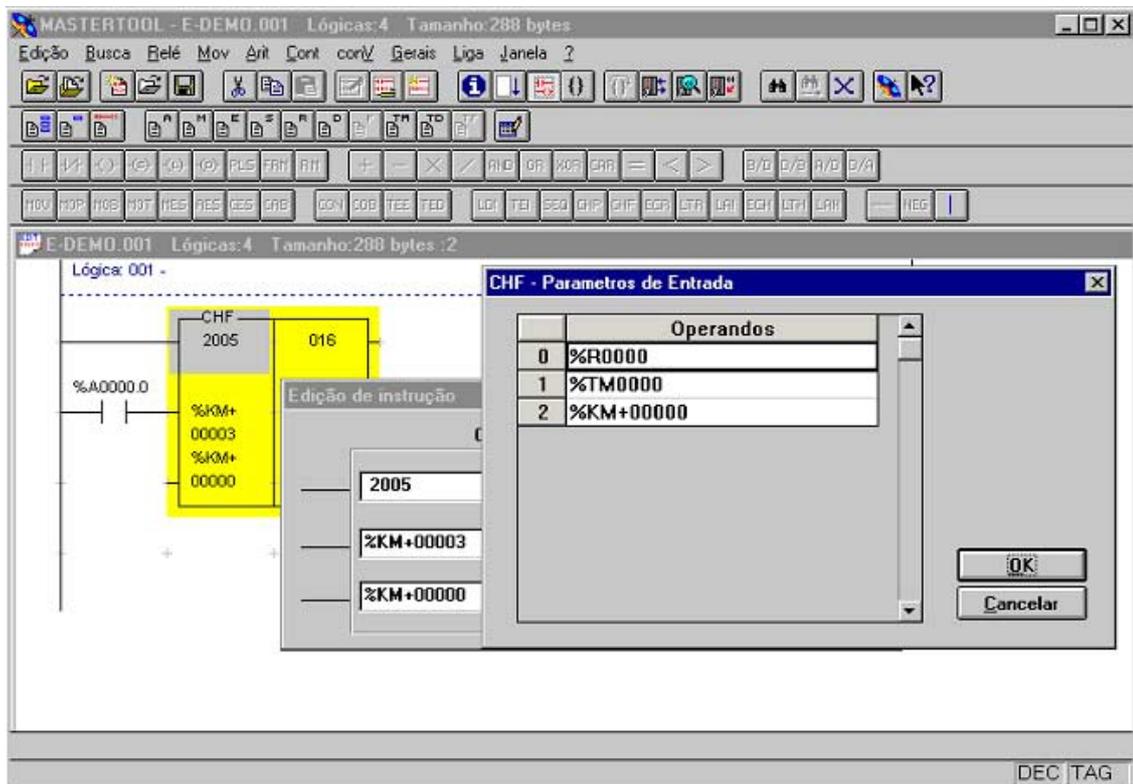



Figura 6-3 Parâmetros de Entrada do Módulo F –2005.016 (1)

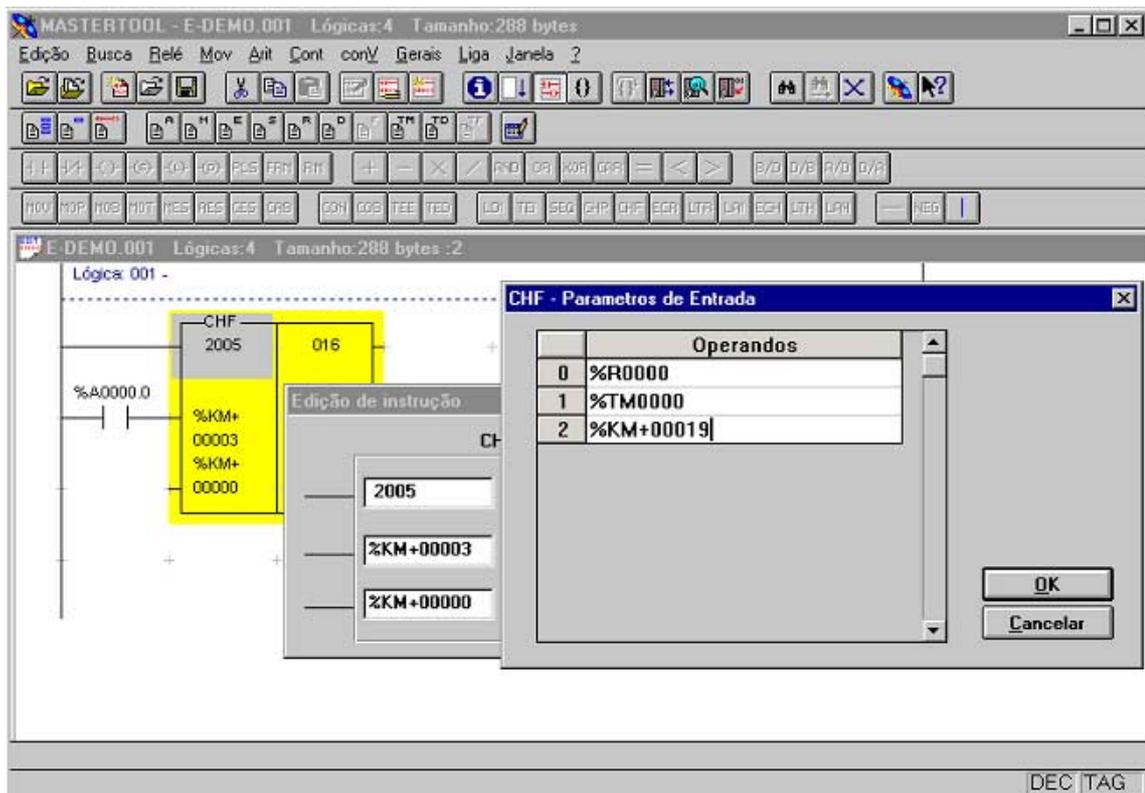



Figura 6-4 Parâmetros de Entrada do Módulo F –2005.016 (2)

Utilizando os Programas DEMO

Os programas DEMOs são fornecidos como parte integrante do AL-2005 e tem como objetivo servir de base para o desenvolvimento de novas aplicações.

O desenvolvimento de novos programas aplicativos deve, preferencialmente, utilizar um destes exemplos como modelo, já que nele estão inclusas as chamadas de todas as ferramentas necessárias para compilar, depurar e gerar programas executáveis no processador AL-2005.

Os dois exemplos foram desenvolvidos visando utilizarem recursos distintos disponíveis no ambiente, de modo a oferecer uma gama maior de características demonstradas.

Utilizando o recurso de menus configuráveis do compilador Borland C++ é possível chamar as seguintes ferramentas, já com suas linhas de comando convenientemente acertadas:

- o depurador Turbo Debugger
- o programa LOCATE para relocar aplicações a serem depuradas remotamente
- o programa AL-3860 de carga de aplicações na memória Flash EPROM do AL-2005

A programação dos menus configuráveis do compilador Borland C++ fica armazenada nos arquivos de projeto .PRJ. Sugere-se portanto, que, ao se criar uma nova aplicação, apenas sejam copiados os arquivos DEMO.PRJ e DEMOTD.PRJ para outros nomes (como PROG.PRJ e PROGTD.PRJ, por exemplo), alterando-se os parâmetros necessários, mas mantendo-se a programação dos menus, o que possibilita a chamada de todas as ferramentas necessárias a partir do compilador.

Verifique qual das aplicações demo melhor se aplica ao desenvolvimento desejado.

Os parâmetros passados ao depurador Turbo Debugger e ao programa LOCATE devem ser configurados corretamente. Para visualizar os parâmetros de configuração disponíveis em cada um deles, deve-se invocar o Turbo Debugger com o parâmetro /h (“TD /h”) e o LOCATE sem nenhum parâmetro (“LOCATE”).

Para tanto, configure o projeto para os diretórios de instalação do Borland C e de sua aplicação.

As figuras a seguir exibem a sequência para ajuste dos diretórios – menu Options / Directories - e dos comandos – menu Options / Transfer – no ambiente do Borland C:

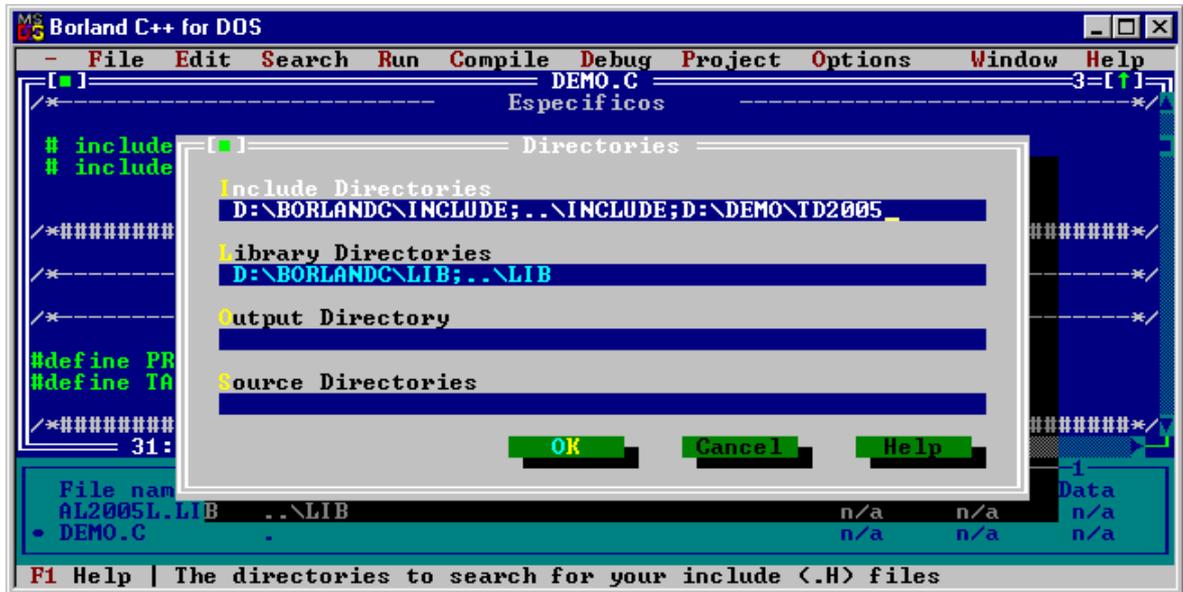



Figura 6-5 Ajuste de Diretórios

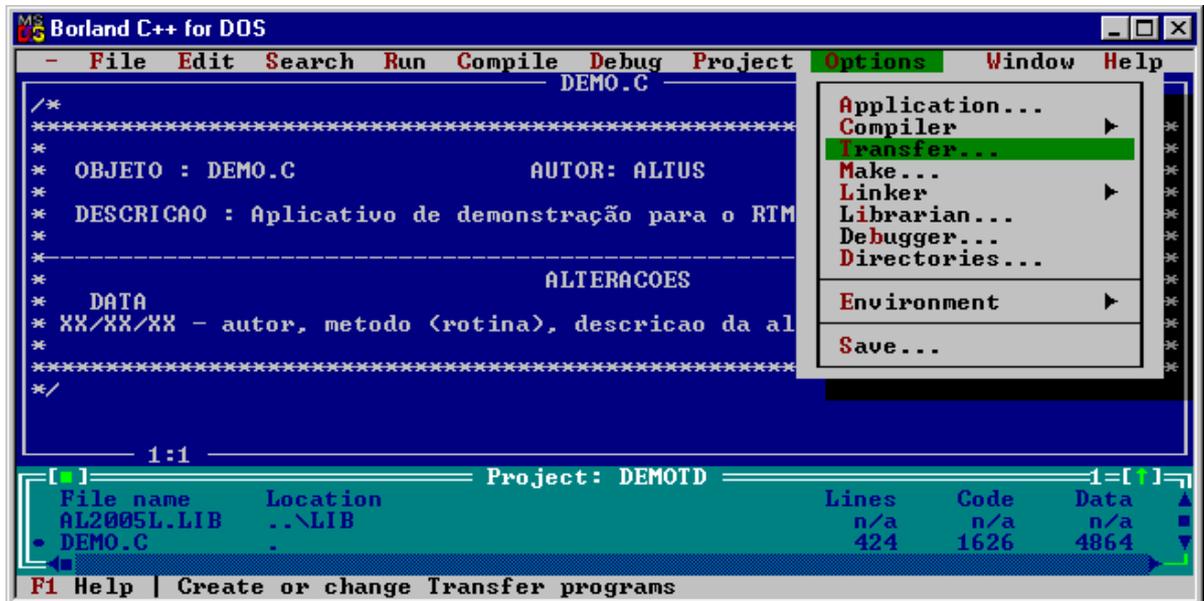



Figura 6-6 Ajuste de Comandos (1)

Figura 6-7 Ajuste do Comando Relocador (1)

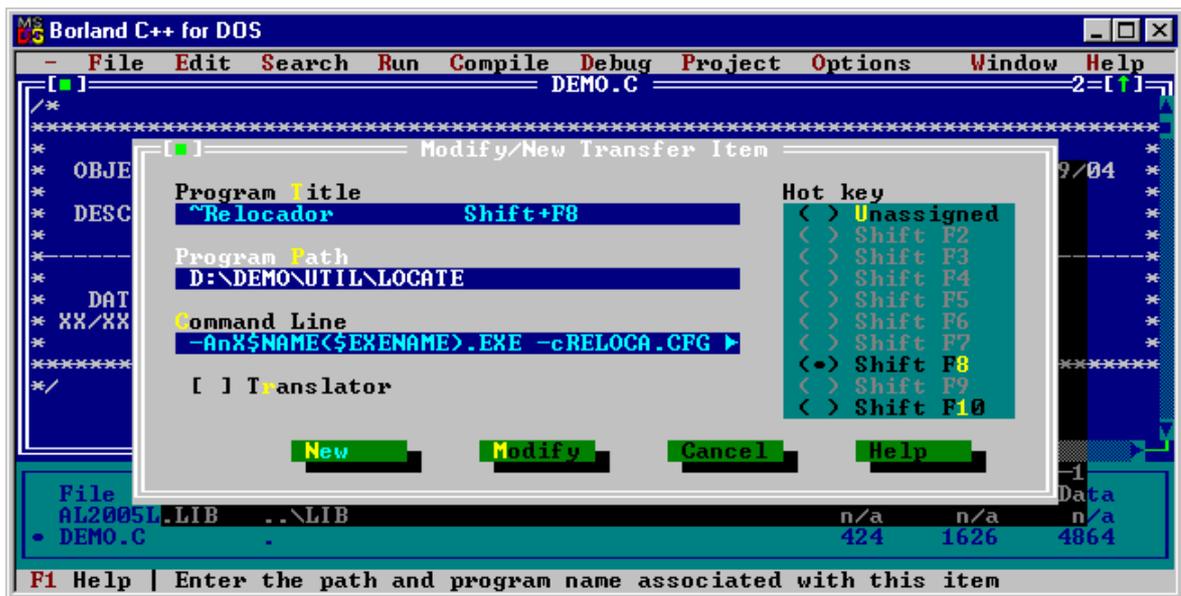



Figura 6-8 Ajuste do Comando Relocador (2)

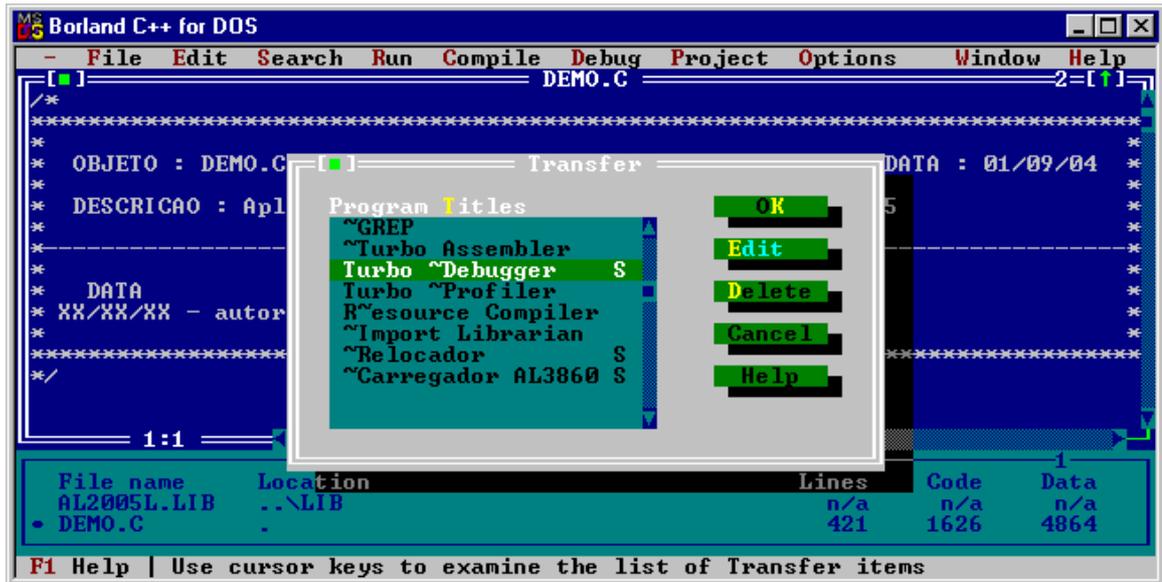



Figura 6-9 Ajuste do Comando Turbo Debugger (1)

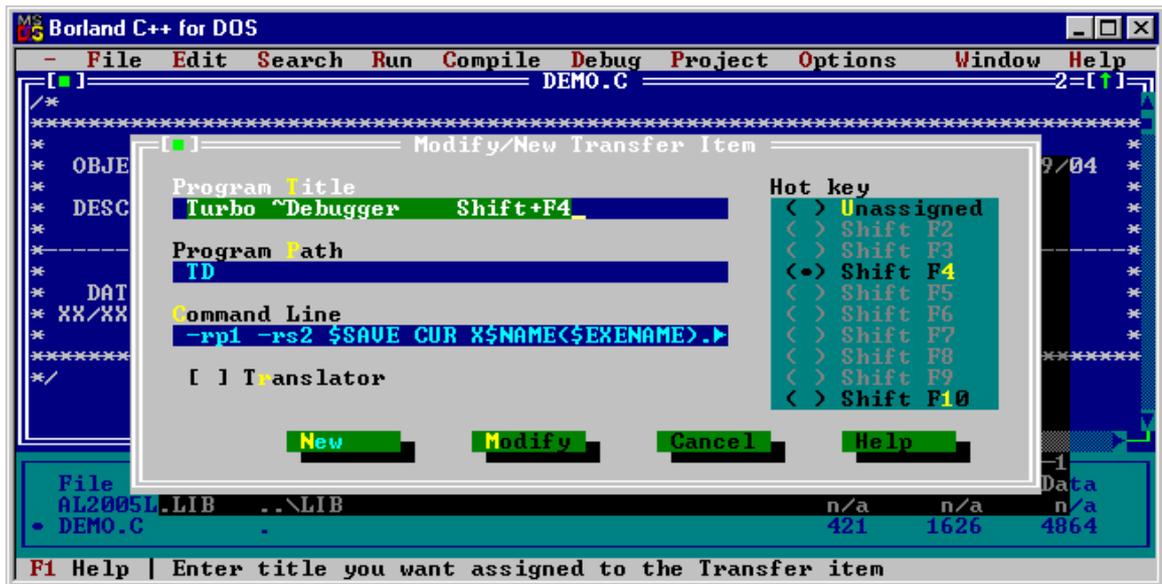



Figura 6-10 Ajuste do Comando Turbo Debugger (2)

Figura 6-11 Ajuste do Comando Carregador AL-3860 (1)

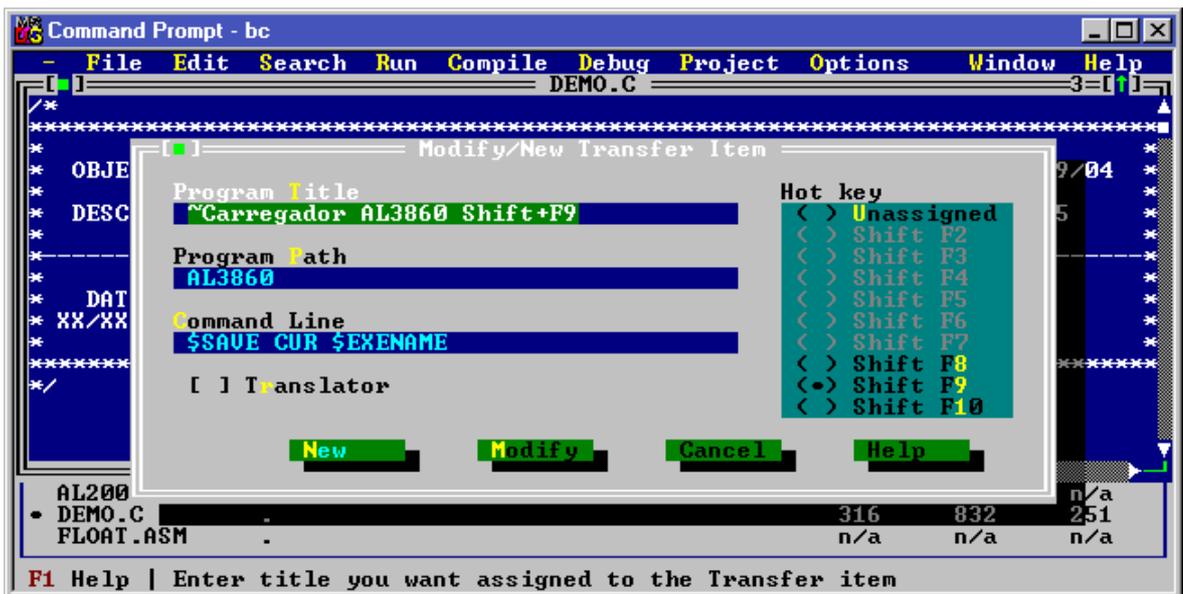



Figura 6-12 Ajuste do Comando Carregador AL-3860 (2)

As etapas do desenvolvimento de um programa aplicativo, sob o ponto de vista do ambiente, ou das ferramentas de desenvolvimento são detalhadas a seguir:

- escrita do programa aplicativo
- depuração do programa
- geração do programa executável
- carga do programa executável na Flash EPROM do processador AL-2005

O AL-2005 executa em paralelo vários aplicativos. Cada aplicativo pode ser seqüencial, sendo uma única tarefa ou multitarefa.

São características da programação seqüencial:

- programação simplificada em relação a criar um aplicativo com várias tarefas
- utiliza poucas rotinas da API do AL-2005

São características da programação multitarefa:

- mais complexo, exigindo maior experiência do projetista
- utiliza maior quantidade de rotinas da API do AL-2005 para efetuar sincronização entre as tarefas
- desenvolvimento mais específico (menos portável)
- permite subdividir o processamento em trechos de código independentes que cooperam para atingir um objetivo comum

Descrição do Programa DEMOCOM

O programa exemplo fornecido no diretório DEMOCOM tem por objetivo demonstrar a utilização de funções da API responsáveis por acesso às interfaces serias, além de demonstrar a criação e sincronização de múltiplas tarefas em uma aplicação do processador AL-2005.

O arquivo DEMO.C, fornecido pelo DEMOCOM implementa duas tarefas que são executadas concorrentemente. São elas:

- driver de comunicação para protocolo ALNET I, que interpreta APENAS comandos de monitoração de operandos do CP:
    - na posição TM0[2] deve ser especificado o canal serial do AL-2005 utilizado:
0 para COMA (configuração padrão)
1 para COMB
    - na posição TM0[4] aparece um código para a velocidade de comunicação desejada:
0 para 38400 bps
1 para 19200 bps
2 para 9600 bps (configuração padrão)
3 para 4800 bps
4 para 2400 bps
5 para 1200 bps
6 para 600 bps
7 para 300 bps
8 para 150 bps
- escrita contínua na tabela TM3 do CP:
    - na posição TM3[0] aparece o número de posições da TM3 correntemente escritos, que varia de 256 a 4, seqüencialmente
    - a posição TM3[1] contém o número da iteração para um dado tamanho de transferência especificado em TM3[0] (no total, são executadas 1024 iterações a cada vez para cada valor diferente em TM3[0])

O arquivo DEMO.C inclui comentários no seu código fonte visando facilitar seu entendimento.

Descrição do Programa DEMOCALC

O programa exemplo fornecido no diretório DEMOCALC possui o arquivo DEMO.C, o qual implementa apenas uma função destinada ao cálculo de operações sequencias – laço – que envolvam operações matemáticas complexas.

O projeto contido no exemplo DEMOCALC inclui uma biblioteca de processamento de dados com padrão descrito pela norma IEEE 754 para operações envolvendo dados com ponto flutuante. Esta biblioteca define o padrão *pseudo float* ao invés de dados tipo float. Assim, se obtem uma melhor performance do processador AL-2005 no processamento de dados envolvendo ponto flutuante.

O programa exemplo foi desenvolvido para efetuar as operações aritméticas soma, subtração, multiplicação e divisão envolvendo valores com ponto flutuante. Para tanto, define apenas um esqueleto envolvendo a configuração da aplicação através da função HookPLC e uma rotina principal responsável por identificar a operação e processar a ação solicitada.

Com o uso da função HookPLC é possível configurar-se o operando %M inicial da sequência de dados trocados entre a aplicação no AL-2005 e o programa aplicativo, ao invés de pré-defini-los, como feito no exemplo DEMOCOM.

Através destes parâmetros passados ao coprocessador pela UCP, é executado o cálculo necessário segundo a operação solicitada.

No exemplo, conforme definido na tabela de configuração da aplicação:

- nas posições TM0[2] e TM0[3] devem ser especificados o código da aplicação e a versão. No exemplo:
  TM0[2] = 9999   TM0[3] = 100
- na posição TM0[4] é definido o endereço do operando %M para retorno da configuracao efetuada. No exemplo :
  TM0[4] = 1   o que identifica %M0001. Se %M0001 = –321768 então a aplicação foi configurada com sucesso.
- na posição TM0[5] deve ser especificado o endereço inicial do operando M origem da área de troca de dados. No exemplo:
  TM0[5] = 10   o que identifica %M0010

.

Na área definida no exemplo como iniciando em %M0010, são efetuadas as trocas de dados utilizadas no cálculo. As seguintes informações são definidas a partir deste operando:

- %M0010: controle de habilitação do cálculo
  1 = habilita cálculo para ser efetuado
  0 = identifica que operacao foi efetuada e o resultado pode ser lido pelo ladder
- %M0011: identifica tipo de operacao aritmética
  1 = soma
  2 = subtração
  3 = multiplicação
  4 = divisão
- %M0012: reservado
- %M0013: valor do parâmetro 1 (inteiro)
- %M0014: reservado
- %M0015: valor do parâmetro 2 (inteiro)
- %M0016: reservado
- %M0017: resultado da operação (inteiro)
- %M0018: status do cálculo.
  –321768 = Identifica se operação foi válida.

O resultado do cálculo é armazenado em %M0017 e representa o valor obtido pela operação de P1 + P2, P1 – P2, P1 * P2 ou P1 / P2, e está disponível para uso pelo programa ladder quando %M0010 passar para o valor 0 e %M0018 for –32768..

Da mesma forma que o projeto anterior, DEMO.C inclui comentários no seu código fonte visando facilitar seu entendimento.

Compilando os Programas DEMOCOM e DEMOCALC

Os projetos modelos DEMO são compilados no próprio ambiente de edição do Borland C. Através da opção de menu Compile. O diretório do compilador Borland C deve estar definido no *path* do sistema operacional.

As figuras a seguir mostram o resultado da compilação do projeto.

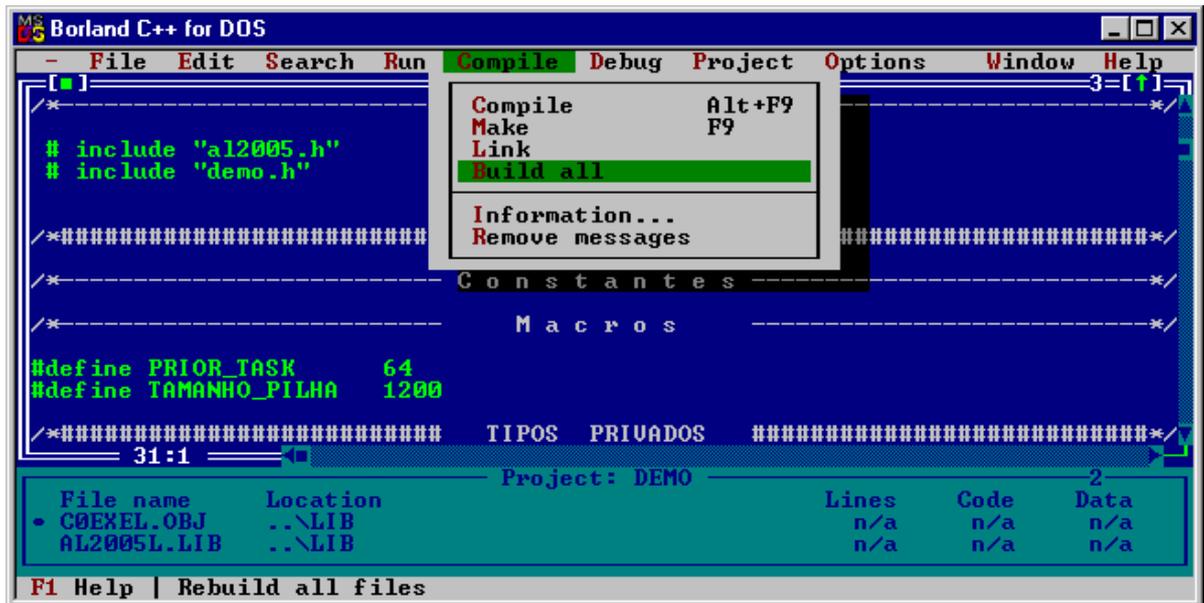



Figura 6-13 Menu Compile

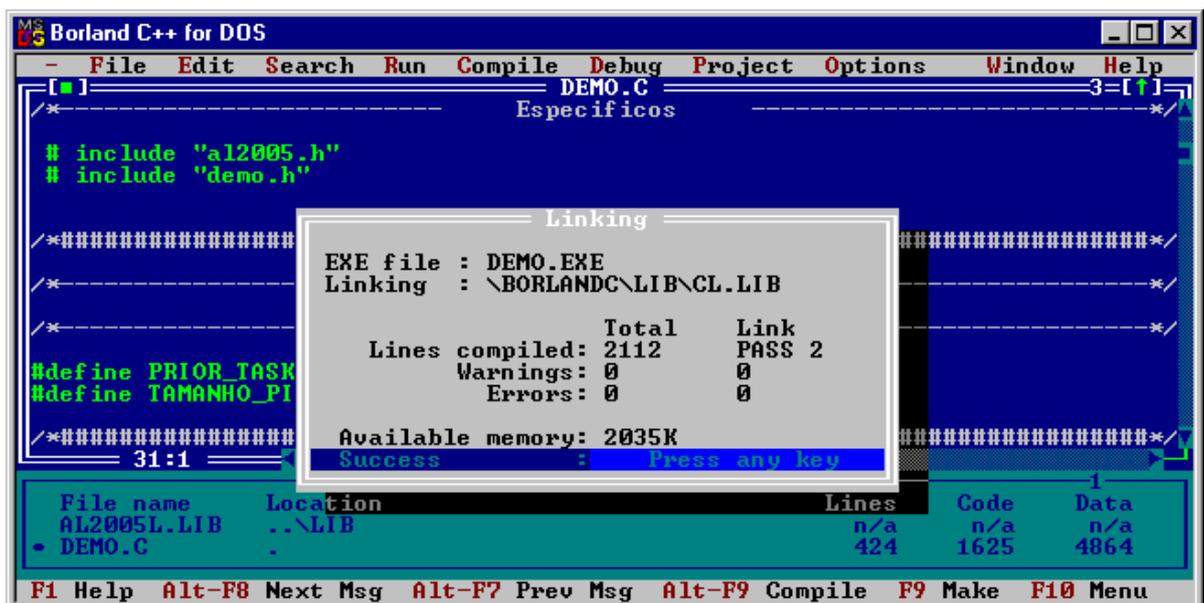



Figura 6-14 Geração do Código Executável

Depuração dos Programas DEMO

Versão para Depuração

Para gerar o aplicativo de demonstração a ser depurado via Turbo Debugger deve ser utilizado o arquivo de projeto do BorlandC DEMOTD.PRJ.

Dentro do ambiente do compilador, pode-se abrir uma janela com o arquivo DEMO.C, de modo a poder editá-lo conforme desejado. Para tanto, basta posicionar o cursor da janela de projeto sobre DEMO.C e teclar ENTER, conforme mostra a figura 6-15:

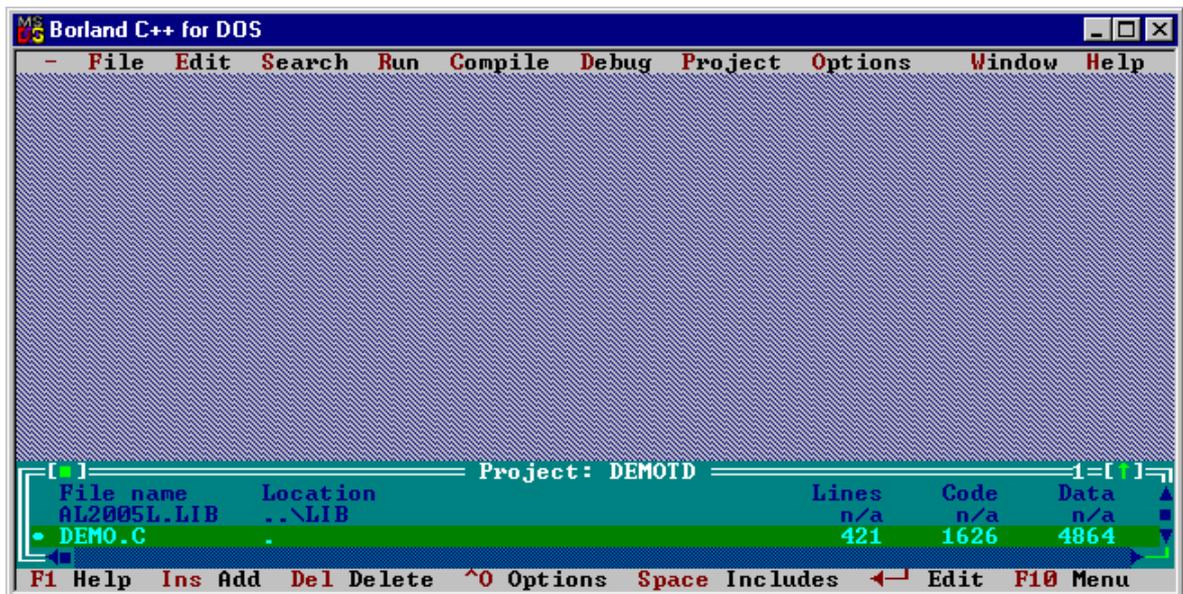



Figura 6-15 – Projeto DEMOTD.PRJ

Para compilar o aplicativo e gerar o executável correspondente, basta teclar F9 (que eqüivale ao menu COMPILE, comando MAKE). A figura 6-16 mostra o menu de compilação:

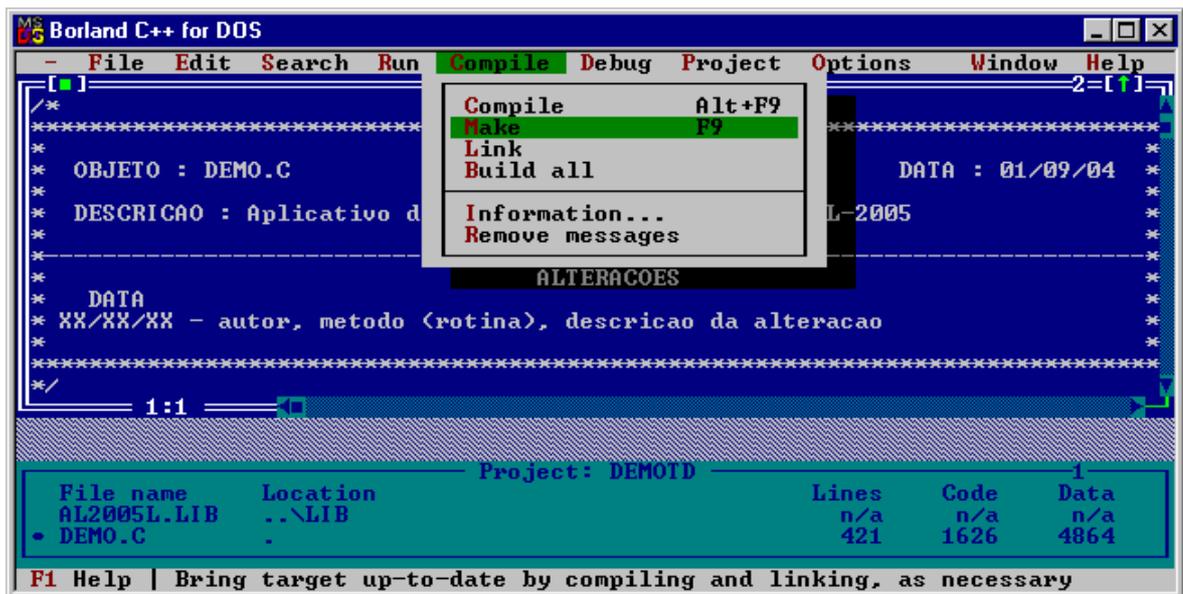



Figura 6-16 – Compilando o Projeto DEMOTD.PRJ

Na sequência deve ser executado o relocador para aplicações a serem depuradas remotamente. Para tanto tecle SHIFT+F8 (que corresponde ao menu de sistema comando RELOCADOR), quando é executado o LOCATE da Paradigm, que utiliza o arquivo de configuração RELOCA.CFG e gera o arquivo XDEMOTD.EXE, previamente configurado.na opção Options/ Transfer .

Antes de iniciar a depuração do aplicativo de demonstração, deve-se carregar no AL-2005 o programa depurador remoto TD2005.EXE, a ser utilizado em conjunto com o Turbo Debugger. Para tanto, tecle SHIFT+F9 (que corresponde ao menu de sistema CARREGADOR AL-3860).

No programa de carga AL3860:

1. Selecione o canal de comunicação a ser utilizado, no botão Porta Serial. A figura 6.17 mostra o menu:

Figura 6-17 – Seleção do Canal Serial

2. Coloque o AL-2005 em estado programação
3. Se for o caso, escolha o comando APAGA, que apaga todos os módulos carregados na memória Flash EPROM do AL-2005. A figura 6-18 mostra a tela do carregador:

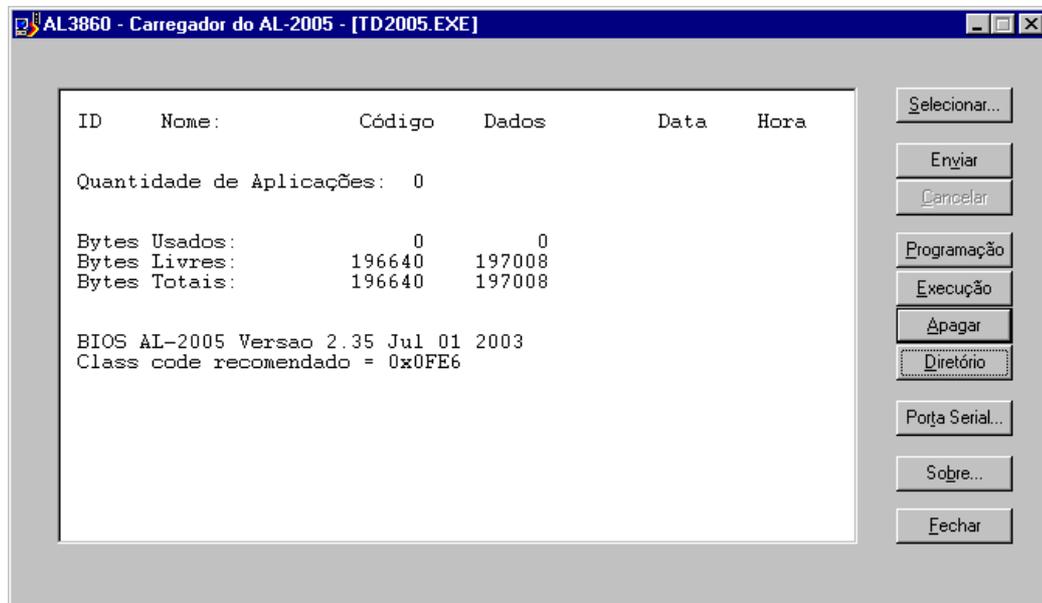



Figura 6-18 – Modo Programação do AL-2005

4. Selecione e envie o arquivo TD2005.EXE para o AL-2005

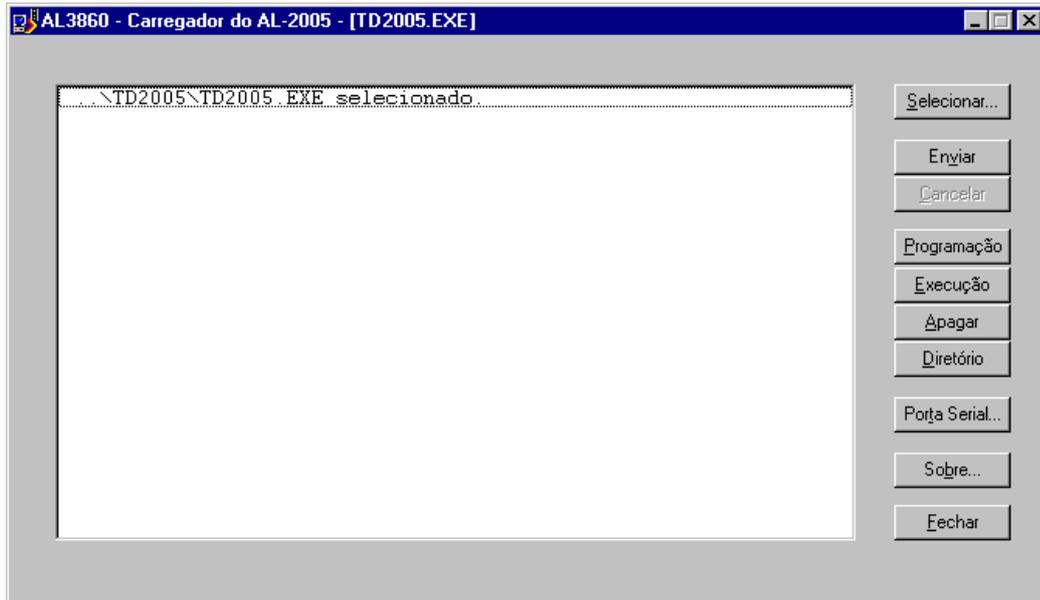



Figura 6-19 – Seleção do Arquivo de Depuração

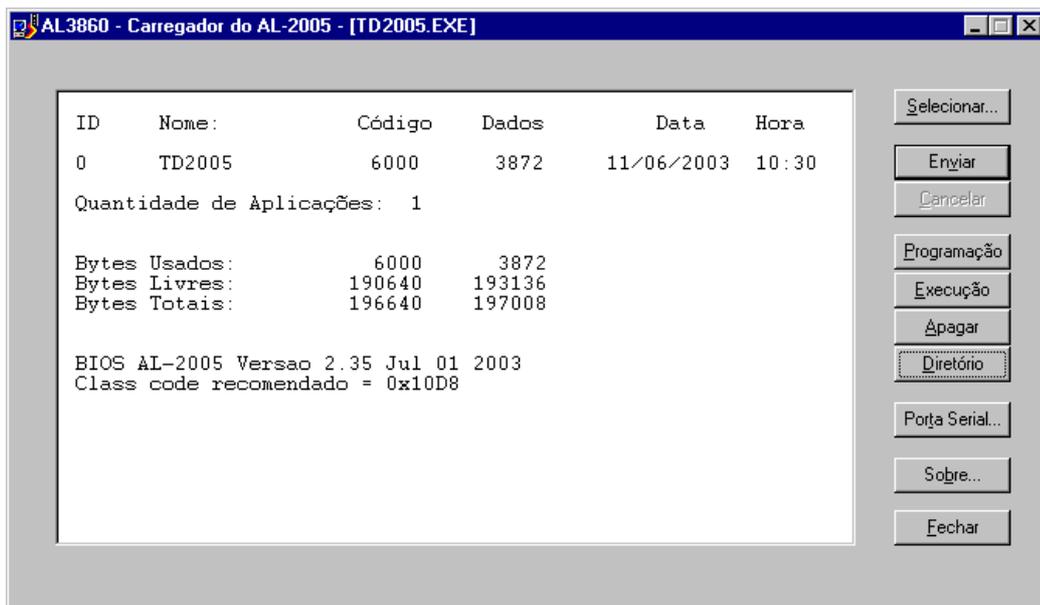



Figura 6-20 – Diretório de Módulos do AL-2005

5. Coloque o AL-2005 em estado de execução

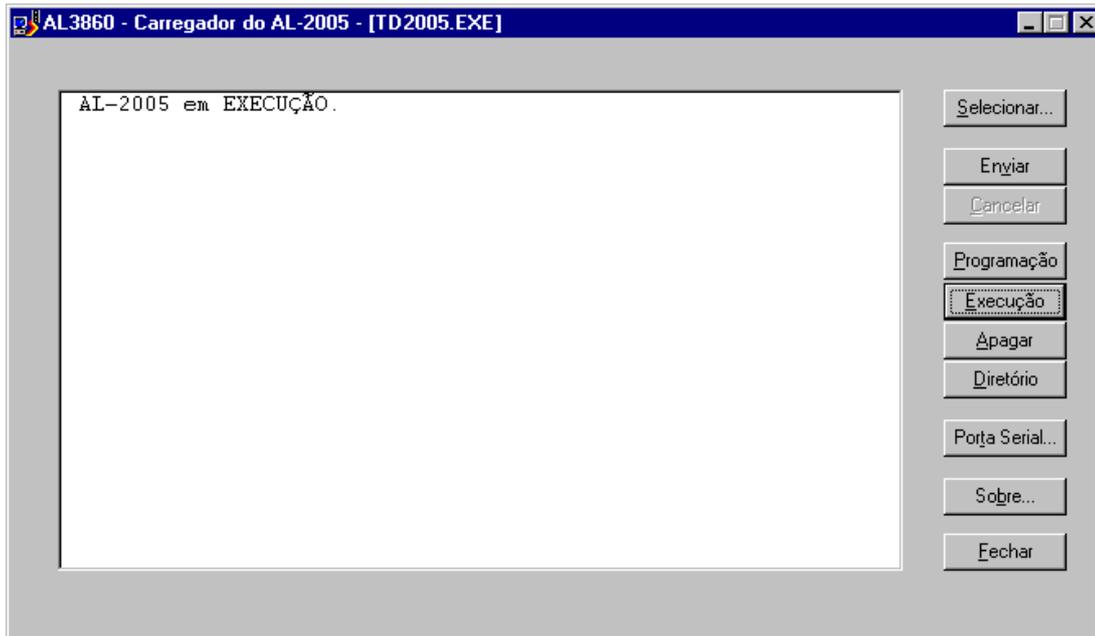



Figura 6-21 – Modo Execução do AL-2005

6. Selecione o botão FECHAR e retorne ao ambiente do compilador

Para depurar o aplicativo de demonstração, tecle SHIFT+F4 (correspondente ao menu de sistema TURBO DEBUGGER) a fim de ativar o Turbo Debugger. Quando o depurador entra no ar, aparece a mensagem: "Program out of date on remote, send over link?". Responda com "Yes" e aguarde o término da carga do aplicativo no AL-2005. A partir deste momento pode ser iniciada a sessão de depuração do aplicativo de demonstração, que pode ser encerrada teclando-se ALT+X.

As figura 6.22 e 6.23 exibem este processo, onde se indentifica o cursor para execução passo a passo da aplicação no AL-2005..

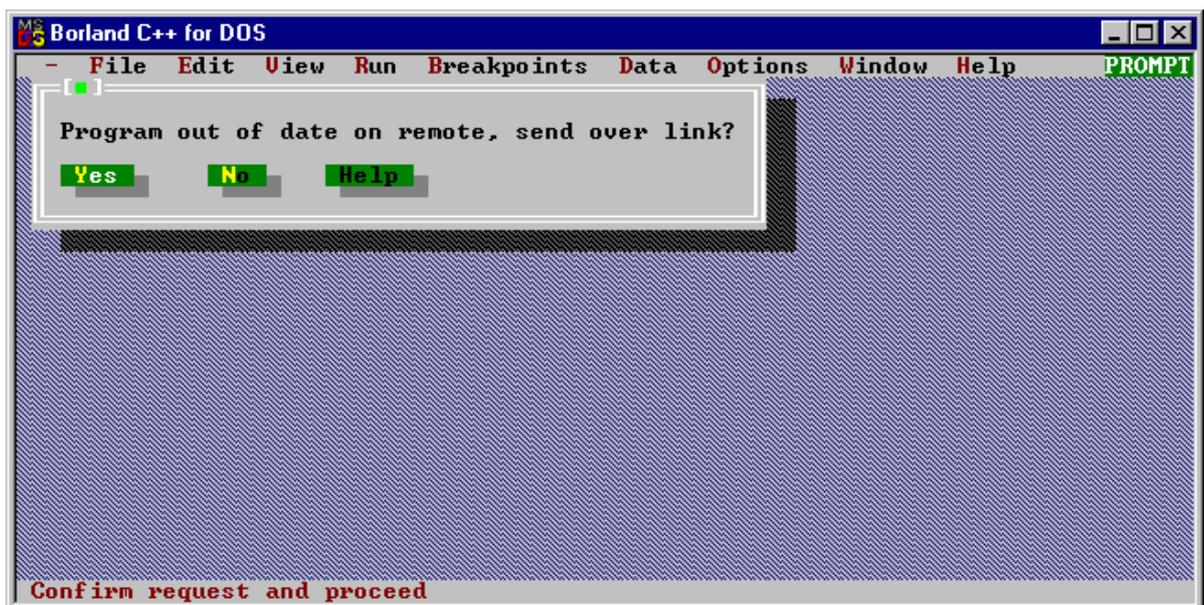



Figura 6-22– Envio do Programa a ser Depurado no AL-2005

```
Borland C++ for DOS
 -  File  Edit  View  Run  Breakpoints  Data  Options  Window  Help     READY
[■]=Module: DEMO File: DEMO.C 414                                  1=[↑][↓]
 }

 void huge main (void)
 {
         CreateTask (&DescrAlnetI, &TaskID_AlnetI);
         StartTask(TaskID_AlnetI);
         CreateTask (&DescrWriteCP, &TaskID_WriteCP);
         StartTask(TaskID_WriteCP);

         Wait();
 }
 Watches                                                            2
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

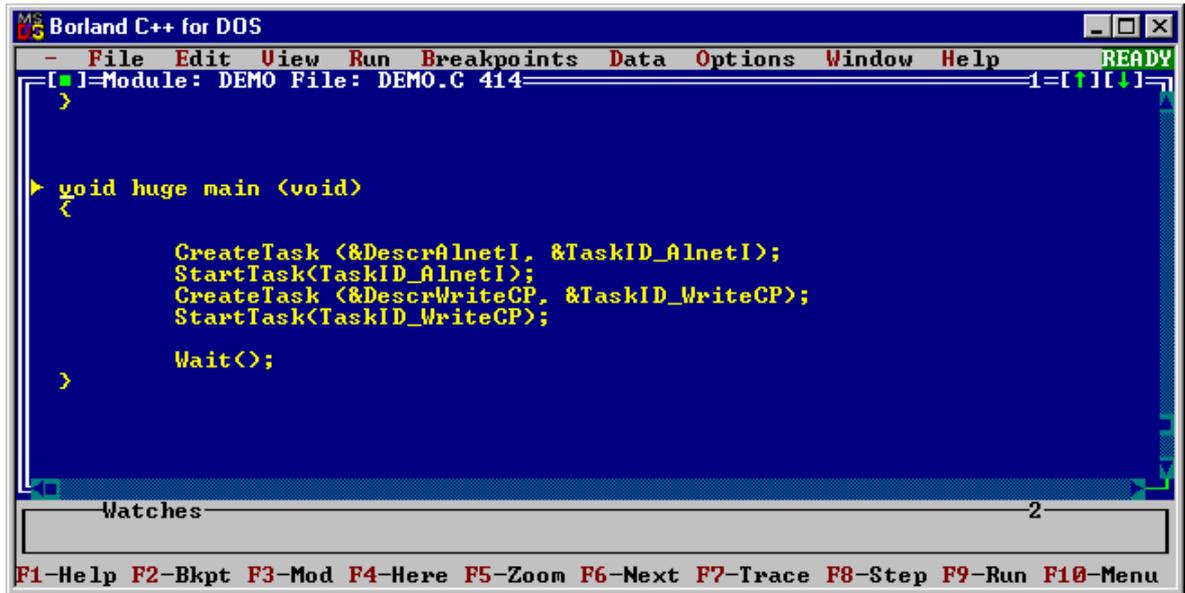



Figura 6-23– Depuração do Programa

Para sair do ambiente do compilador Borland C++, tecle ALT+X.

> ATENÇÃO:
> Conforme já citado, o número da aplicação para uso com o depurador deve ser **19**, e este número deve ser utilizado na definição da chamada à função F-2005.016 (constante memória passado como terceiro parâmetro para a função).

O tamanho do arquivo executável (aplicação) pode não equivaler a soma da memória de código e de dados da aplicação, após a carga no AL-2005, visualizada pelo carregador AL-3860. A diferença que pode haver se deve aos seguintes fatores:

- o header (cabeçalho) do executável não é transferido para o AL-2005
- a tabela de símbolos do executável não é transferida para o AL-2005
- a área de dados no executável pode estar compactada
- variáveis pré-inicializadas ocupam memória de código e de dados no AL-2005, enquanto que no executável somente de código

> O comando de *Diretório* do carregador AL-3860 apresenta entre outros, o dado *Class CODE recomendado = 0xXXXX*. O valor indicado representa a primeira posição de memória livre na RAM do AL-2005 (endereço de parágrafo), e equivale a diretiva CLASS CODE = XXXX do arquivo de configuração do relocador da Paradigm.

Versão para Carga em Flash EPROM

Para gerar o aplicativo de demonstração demonstração para ficar residente na Flash EPROM utilize o arquivo de projeto do BorlandC DEMO.PRJ.

O fonte DEMO.C pode ser alterado conforme desejado. Para tanto pode-se abrir uma janela com o arquivo DEMO.C, posicionando-se o cursor da janela de projeto sobre DEMO.C e digitando ENTER. Para compilá-lo, basta teclar F9, equivalente ao menu COMPILE, comando MAKE.

Para carregar o executável gerado no AL-2005, basta teclar SHIFT+F9 (que corresponde ao menu de sistema CARREGADOR AL-3860).

No programa de carga AL3860:

1. Coloque o AL-2005 em estado programação
2. Se for o caso, escolha o comando APAGA, que apaga todos os módulos carregados na memória Flash EPROM do AL-2005
3. Selecione e envie o arquivo DEMO.EXE para o AL-2005

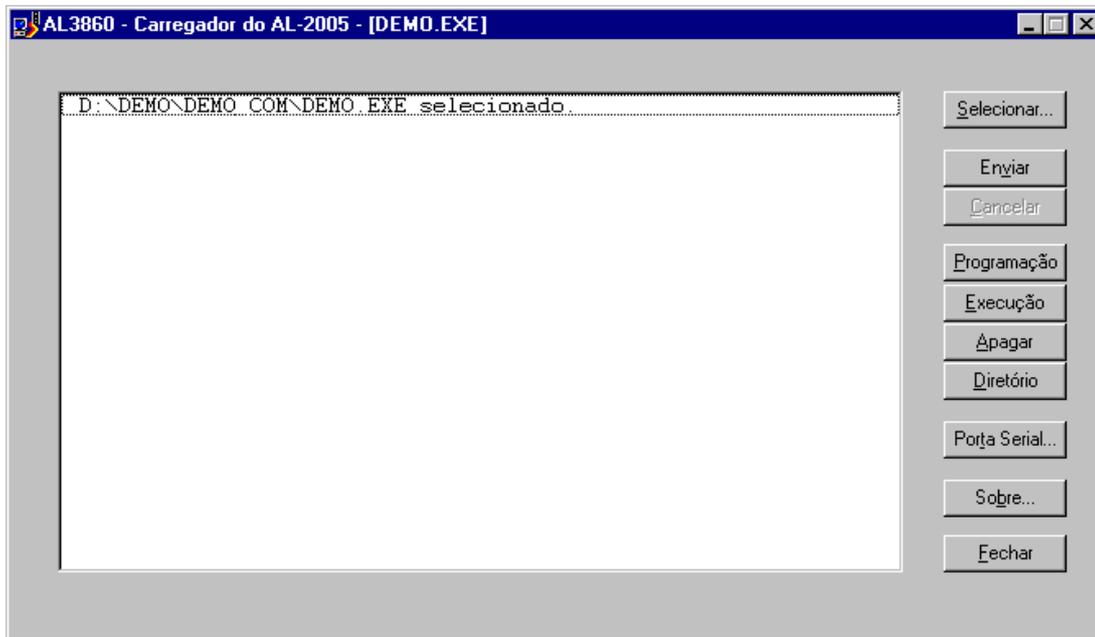



Figura 6-24 – Seleção do Arquivo Final

4. Coloque o AL-2005 em estado de execução
5. Selecione o botão FECHAR e retorne ao ambiente do compilador

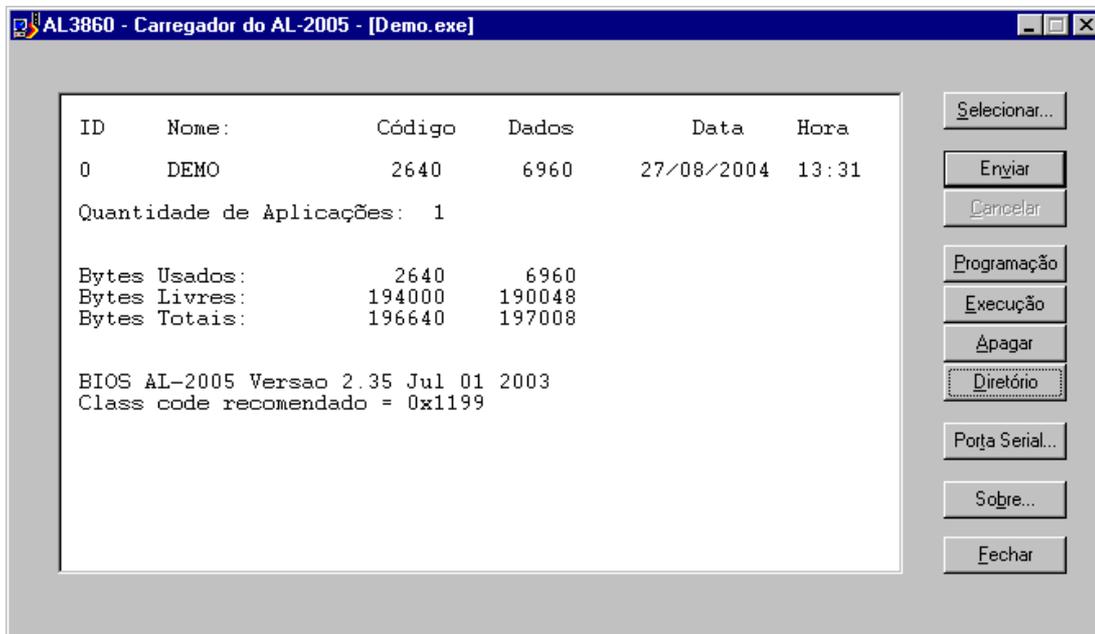



Figura 6-25 – Diretório de Aplicações no AL-2005

Para sair do ambiente do compilador Borland C++, tecle ALT+X.

7. Instalação

Este capítulo descreve os procedimentos e cuidados necessários para a instalação do Processador Multitarefa AL-2005 e dos arquivos que acompanham o produto. As recomendações apresentadas devem ser seguidas para o correto funcionamento do equipamento.

A figura 7-1 mostra o Processador Multitarefa, destacando-se os elementos de conexão e sinalização referenciados nas próximas seções.

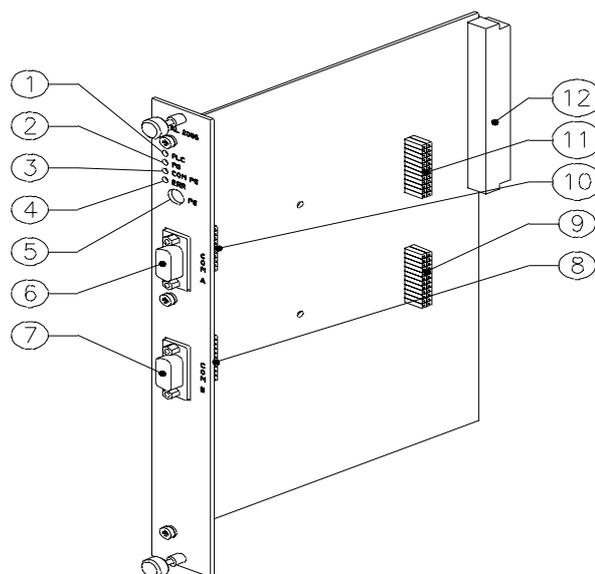



Figura 7-1 Visão Geral do Processador Multitarefa AL-2005

- 1, 2, 3, 4 - LEDs de estados
- 5 - Canal serial dedicado para carga de programa e depuração
- 6 - Conector DB9 do canal "A" de comunicação
- 7 - Conector DB9 do canal "B" de comunicação
- 8, 9 - Conectores para módulo serial do canal "B"
- 10, 11 - Conectores para módulo serial do canal "A"
- 12 - Conector do Processador Multitarefa AL-2005/RTMP ao barramento da UCP AL-2003 ou AL-2004

Instalação Mecânica e Elétrica

O Processador Multitarefa AL-2005 é um módulo utilizado com o sistema das UCPs AL-2003 ou AL-2004. Desta forma seus procedimentos de instalação mecânica e elétrica são descritos no capítulo de Instalação do Manual de Utilização da UCP correspondente.

Conexões Gerais

Interfaces Seriais

O Processador Multitarefa AL-2005 possui um canal serial dedicado RS-232C para carga de programa e depuração de aplicativos e dois conectores seriais DB9 para conexão entre o Processador Multitarefa AL-2005 e equipamentos quaisquer.

Para a carga de programa ou depuração de aplicativos é necessário conectar um microcomputador padrão IBM PC® ao Processador Multitarefa AL-2005 (soquete PG do painel frontal) através do cabo AL-1340 ou AL-1327, conforme o tipo de conector.

A figura 7-2 mostra a conexão do canal serial COM A (conector fêmea DB9) do Processador Multitarefa AL-2005 a um equipamento qualquer.

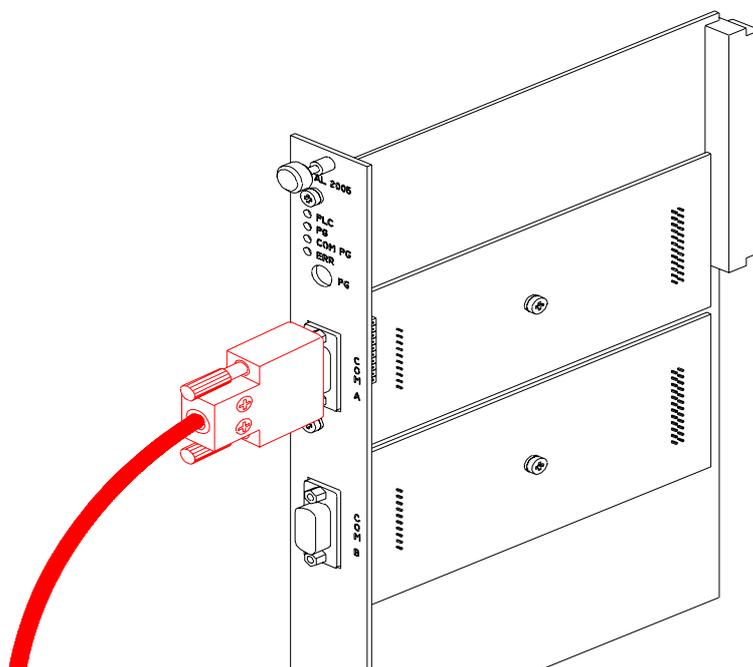



Figura 7-2 Conexão do Canal Serial

Os conectores COM A e COM B apenas realizam a ligação física entre os cabos do meio externo com os módulos de interface acoplados ao Processador Multitarefa AL-2005. Estes módulos são opcionais, oferecendo diferentes tipos de interfaces seriais:

- AL-2405/485I: é um módulo serial que implementa o padrão de comunicação RS-485
- AL-2405/232: é um módulo serial que implementa o padrão RS-232C

ATENÇÃO: Antes de conectar o processador multitarefa AL-2005 a qualquer outro equipamento com canal serial, é indispensável que ambos possuam um ponto de aterramento em comum.

Instalação dos Módulos Seriais

A figura a seguir mostra como deve ser executado o acoplamento de módulos seriais ao Processador Multitarefa AL-2005.

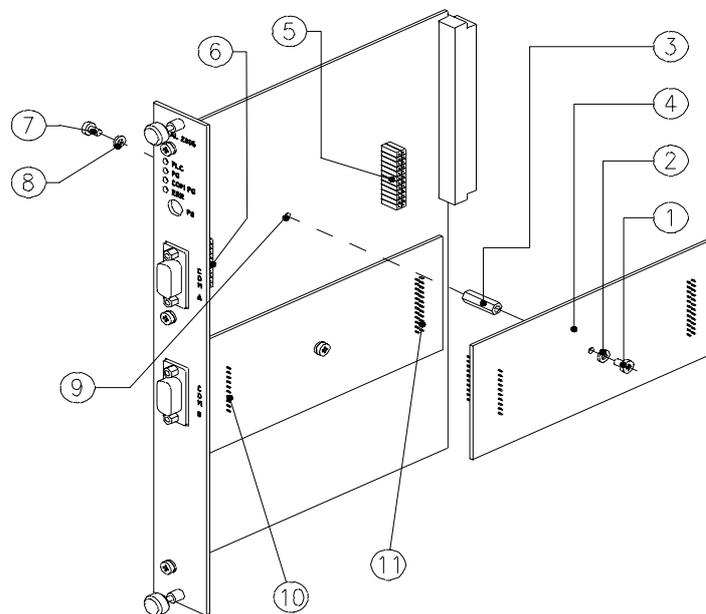



Figura 7-3 Conexão do Módulo Serial

Para realizar a instalação do módulo serial devem ser seguidos os seguintes passos:

1. Parafusar o espaçador (3) na placa AL-2005 fixando com o parafuso (7) e arruela (8) pelo orifício (9)

2. Encaixar a placa do módulo serial nos conectores 5 e 6 (ou 10 e 11) na placa AL-2005

3. Verificar se todos os pinos encaixam corretamente nos seus respectivos conectores

4. Parafusar a placa do módulo serial através do parafuso (1) e arruela (2) pelo orifício (4)

Instalação do CD

O CD-ROM contém um aquivo denominado AL2005.EXE.

Para instalação, execute o instalador e siga os procedimentos exibidos na tela

.

8. Manutenção

Ao longo da operação ou funcionamento do sistema, algumas anormalidades podem ser eventualmente encontradas pelo usuário. Os itens a seguir apresentam as anormalidades mais comuns e dão instruções sobre os procedimentos a serem tomados em cada caso.

Diagnósticos

Diagnósticos são mensagens que o sistema envia ao usuário relatando anormalidades, sejam elas falhas, erros ou modos de operação.

O AL-2005 possui quatro LEDs no seu painel frontal para indicar diferentes modos de operação, auxiliando também no diagnóstico de possíveis erros.

Suas indicações são apresentadas na tabela a seguir:

LEDs	Significado
ERR aceso e PG piscando	A memória do AL-2005/RTMP está vazia
PG aceso	AL-2005 em estado programação
PLC piscando	AL-2005 em execução e o processador comunicando com a UCP
PLC permanentemente aceso	Possível erro na aplicação (laço repetitivo).
COM PG piscando	AL-2005 comunicando com o computador pela porta serial de programação, durante a carga de aplicação
PLC, PG, COM PG e ERR acesos por alguns instantes	Situação normal de power up (inicialização do AL-2005)
PLC, PG, COM PG e ERR permanentemente acesos	Caso ocorra durante a execução da aplicação, indica erro irrecuperável

Tabela 8-1 - LED DG

Teste das Interfaces Seriais

O programa TSTCOM fornecido juntamente com o módulo AL-2005 tem por por finalidade testar os canais A e B do mesmo.

O software presente no módulo fica constantemente informando pela porta serial a mensagem:

"<cn> AL-2005 com A serial test", no caso do software TSTCOM_A.EXE.

O parâmetro <cn> é um contador livre que informa o número da mensagem enviada.

Para teste do canal B deve ser utilizado o programa TSTCOM_B.EXE.

Para verificação do funcionamento das interfaces seriais devem ser seguidos os seguintes procedimentos:

1- Carga no módulo AL-2005 do software TSTCOM_A.EXE, TSTCOM_B.EXE ou ambos para testar a porta desejada através do programa de carga AL-3860

2- Neste momento inicia-se a comunicação:

- para TSTCOM_A.EXE irá piscar o LED PG
- para TSTCOM_B.EXE irá piscar o LED COM PG

3- Para comunicação com o AL-2005 deve ser utilizado o programa HyperTerminal chamado no seu microcimputador. Os parâmetros de configuração do canal serial são fixos e seguem o padrão do MasterTool conforme a tabela:

- BaudRate: 9600 bps
- Paridade: par
- StopBit: 1 stopbit
- RTS/CTS: sem controle

4- O software irá imprimir na janela do HyperTerminal a mensagem supra citada e, para cada tecla pressionada, será mostrado um conjunto de caracteres correspondentes à esta letra

Caso ainda não ocorra comunicação, certifique-se de que a placa satélite AL-2405 possui o padrão 232 ou 485, conforme sua configuração.

Identificando Problemas

Quando o AL-2005 não responder adequadamente a uma comunicação solicitada, os seguintes itens devem ser verificados se:

- verificar se os LEDs da fonte de alimentação acendem. Em caso negativo, verificar o estado do fusível da mesma
- a fonte está alimentada com a voltagem correta
- verificar se o dimensionamento de correntes do barramento não ultrapassa o máximo especificado nas características técnicas das fontes
- verificar se o processador multitarefa AL-2005 está corretamente encaixado no **bastidor**
- verificar se os módulos opcionais (ex. Interface RS-485 AL-2405/485I) encontram-se corretamente encaixados nos conectores apropriados do Processador Multitarefa AL-2005
- verificar se o cabo de comunicação serial está conectado
- a temperatura ambiente está dentro do especificado

CUIDADO:
A troca de módulos no barramento ou a verificação de suas conexões deve ser realizada com a alimentação principal do sistema desenergizada .

Se nenhum problema for identificado, consulte o Suporte a Clientes Altus.

Manutenção Preventiva

- Deve-se verificar, a cada ano:
    - se os cabos de interligação estão com as conexões firmes, sem depósitos de poeira, principalmente os dispositivos de proteção
    - se o valor de tensão da fonte de alimentação está dentro do padrão especificado para o módulo
- Em ambientes sujeitos a contaminação excessiva, deve-se limpar periodicamente o equipamento, retirando resíduos, poeira, etc

9. Glossário

Algoritmo	Seqüência finita de instruções bem definidas, objetivando à resolução de problemas.
Barramento	Conjunto de sinais elétricos agrupados logicamente com a função de transferir informação e controle entre diferentes elementos de um subsistema.
Baud rate	Taxa com que os bits de informação são transmitidos através de uma interface serial ou rede de comunicação (medido em bits/segundo).
Bit	Unidade básica de informação, podendo estar no estado 0 ou 1.
BT	Sigla para teste de bateria em inglês (battery test).
Buffers, Pool de	Coleção de buffers de dados cujo uso é controlado pelo gerenciador de buffers.
Byte	Unidade de informação composta por oito bits.
Caixa postal	Receptáculo fornecido por cada tarefa para receber suas mensagens. Cada tarefa pode ter até quatro caixas postais.
Canal serial	Interface de um equipamento que transfere dados no modo serial.
Ciclo de varredura	Uma execução completa do programa aplicativo de um controlador programável.
Circuito de cão-de-guarda	Circuito eletrônico destinado a verificar a integridade do funcionamento de um equipamento.
Código comercial	Código do produto, formado pelas letras PO, seguidas por quatro números.
Código de erro	Série de inteiros com sinal usados para indicar condições de erro detectadas pelas funções da biblioteca.
Controlador programável	Também chamado de CP. Equipamento que realiza controle sob o comando de um programa aplicativo. É composto de uma UCP, uma fonte de alimentação e uma estrutura de E/S.
CP	Veja controlador programável.
Default	Valor predefinido para uma variável, utilizado em caso de não haver definição.
Depuração	Testes para determinação do correto funcionamento do produto, procurando os possíveis erros de um sistema.
Diagnóstico	Procedimento utilizado para detectar e isolar falhas. É também o conjunto de dados usados para tal determinação, que serve para a análise e correção de problemas.
Download	Carga de programa ou configuração no CP.
E/S	Veja entrada/saída.
E2PROM	Memória não-volátil, que pode ser apagada eletricamente.
EIA RS-485	Padrão industrial (nível físico) para comunicação de dados.
Endereço de módulo	Endereço pelo qual o CP realiza acessos a um determinado módulo de E/S.
Entrada/saída	Também chamado de E/S. Dispositivos de E/S de dados de um sistema. No caso de CPs, correspondem tipicamente a módulos digitais ou analógicos de entrada ou saída que monitoram ou acionam o dispositivo controlado.
Envelope	Estrutura de dados privada usada pelo sistema operacional para passar uma mensagem para uma tarefa.
EPROM	Significa Erasable Programmable Read Only Memory. É uma memória somente de leitura, apagável e programável. Não perde seu conteúdo quando desenergizada.
ER	Sigla usada para indicar erro nos LEDs.
Escravo	Equipamento ligado a uma rede de comunicação que só transmite dados se for solicitado por outro equipamento denominado mestre.
Flash EPROM	Memória não-volátil, que pode ser apagada eletricamente.
Frame	Uma unidade de informação transmitida na rede.
Freeze	Em redes PROFIBUS, é o estado da rede quando os dados das entrada são congelados.
Gateway	Equipamento para a conexão de duas redes de comunicação com diferentes protocolos.
Grupo de eventos	Conjunto de 16 eventos cujo acesso e sinalização é controlado pelo gerenciador de eventos.
grupo, Identificador de um	Inteiro sem sinal atribuído a um grupo de eventos pelo sistema operacional para ser usado como seu identificador único.
Hardware	Equipamentos físicos usados em processamento de dados onde normalmente são executados programas (software).
Idle	Um dos estados de uma tarefa no sistema operacional do Processador Multitarefa AL-2005/RTMP . Quando uma tarefa está no estado Idle, ela não está sendo utilizada.
IEC 1131	Norma genérica para operação e utilização de CPs.
Interface	Dispositivo que adapta elétrica e/ou logicamente a transferência de sinais entre dois equipamentos.
Interrupção	Evento com atendimento prioritário que temporariamente suspende a execução de um programa e desvia para uma rotina de atendimento específica
kbytes	Unidade representativa de quantidade de memória. Representa 1024 bytes.
LED	Sigla para light emitting diode. É um tipo de diodo semicondutor que emite luz quando estimulado por eletricidade. Utilizado como indicador luminoso.

Linguagem Assembly Linguagem de programação do microprocessador, também conhecida como linguagem de máquina.

Linguagem de programação Um conjunto de regras e convenções utilizado para a elaboração de um programa.

Linguagem de relés e blocos Altus Conjunto de instruções e operandos que permitem a edição de um programa aplicativo para ser utilizado em um CP.

Lista circular Estrutura de dados da aplicação usada para manter uma lista de 1, 2 ou 4 bytes, com a capacidade de acrescentar e remover elementos tanto no topo quanto na base.

lista, Elemento de uma Um valor de um byte (8 bits), uma palavra (16 bits) ou dupla palavra (32 bits) que pode ser acrescentado ou removido de uma lista circular.

Lógica Matriz gráfica onde são inseridas as instruções de linguagem de um diagrama de relés que compõe um programa aplicativo. Um conjunto de lógicas ordenadas seqüencialmente constitui um módulo de programa.

MasterTool Identifica o programa Altus para microcomputador, executável em ambiente WINDOWS®, que permite o desenvolvimento de aplicativos para os CPs das séries Ponto, Piccolo, AL-2000, Grano e Quark. Ao longo do manual, este programa é referido pela própria sigla ou como programador MasterTool.

Memória imagem Área de memória compartilhada entre o Processador Multitarefa AL-2005/RTMP e a UCP AL-2004 ou AL-2003. A memória imagem contém o valor dos operandos da UCP

Mensagem Doze bytes de informação da aplicação passada pelo sistema operacional em um envelope para uma tarefa.

mensagem, Prioridade de uma Identifica qual entre as quatro caixas postais de uma tarefa deve receber uma mensagem passada pelo sistema operacional.

Menu Conjunto de opções disponíveis e exibidas por um programa no vídeo e que podem ser selecionadas pelo usuário a fim de ativar ou executar uma determinada tarefa.

Mestre Equipamento ligado a uma rede de comunicação de onde se originam solicitações de comandos para outros equipamentos da rede.

Módulo (referindo-se a hardware) Elemento básico de um sistema completo que possui funções bem definidas. Normalmente é ligado ao sistema por conectores, podendo ser facilmente substituído.

Módulo (referindo-se a software) Parte de um programa aplicativo capaz de realizar uma função específica. Pode ser executado independentemente ou em conjunto com outros módulos, trocando informações através da passagem de parâmetros.

Módulo C Veja módulo de configuração.

Módulo de configuração Também chamado de módulo C. É um módulo único em um programa de CP que contém diversos parâmetros necessários ao funcionamento do controlador, tais como a quantidade de operandos e a disposição dos módulos de E/S no barramento.

Módulo de E/S Módulo pertencente ao subsistema de entradas e saídas.

Módulo E Veja módulo execução.

Módulo execução Módulo que contém o programa aplicativo, podendo ser de três tipos: E000, E001 e E018. O módulo E000 é executado uma única vez, na energização do CP ou na passagem de programação para execução. O módulo E001 contém o trecho principal do programa que é executado ciclicamente, enquanto que o módulo E018 é acionado por interrupção de tempo.

Módulo F Veja módulo função.

Módulo função Módulo de um programa de CP que é chamado a partir do módulo principal (módulo E) ou a partir de outro módulo função ou procedimento, com passagem de parâmetros e retorno de valores. Atua como uma sub-rotina.

Módulo P Veja módulo procedimento.

Módulo procedimento Módulo de um programa de CP que é chamado a partir do módulo principal (módulo E) ou a partir de outro módulo procedimento ou função, sem a passagem de parâmetros.

Multitasking Característica de um sistema operacional poder executar várias tarefas de um sistema por intermédio de um método de compartilhamento de UCP

Nibble Unidade de informação composta por quatro bits.

Nó Qualquer estação de uma rede com capacidade de comunicação utilizando um protocolo estabelecido.

Octeto Conjunto de oito bits numerados de 0 a 7.

Operandos Elementos sobre os quais as instruções atuam. Podem representar constantes, variáveis ou um conjunto de variáveis.

PA Ver pontes de ajuste.

PC Sigla para programmable controller. É a abreviatura de controlador programável em inglês.

Preemptivo Método de compartilhamento de UCP por prioridades de execução.

Programa aplicativo É o programa carregado em um CP, que determina o funcionamento de uma máquina ou processo.

Programa executivo Sistema operacional de um controlador programável. Controla as funções básicas do controlador e a execução de programas aplicativos.

Protocolo Regras de procedimentos e formatos convencionais que, mediante sinais de controle, permitem o estabelecimento de uma transmissão de dados e a recuperação de erros entre equipamentos.

RAM Sigla para random access memory. É a memória onde todos os endereços podem ser acessados diretamente de forma aleatória e com a mesma velocidade. É volátil, ou seja, seu conteúdo é perdido quando o equipamento é desenergizado, a menos que se possua uma bateria para a retenção dos

valores.

Ready Um dos estados de uma tarefa no Processador Multitarefa AL-2005/RTMP. Indica que uma tarefa está pronta para executar

Real-Time É uma característica de processos que exigem um tempo de resposta rápido

Recurso Uma entidade, como uma região de memória, cuja posse é controlada pelo gerenciador de recursos.

recurso, Identificador de um Inteiro sem sinal atribuído a um recurso pelo gerenciador de recursos para ser usado como seu identificador único.

Rede de comunicação Conjunto de equipamentos (nós) interconectados por canais de comunicação.

Rede de comunicação determinística Rede de comunicação onde a transmissão e a recepção de informações entre os diversos nós é garantida com um tempo máximo conhecido.

Rede de comunicação mestre-escravo Rede de comunicação onde as transferências de informações são iniciadas somente a partir de um único nó (mestre da rede) ligado ao barramento de dados. Os demais nós da rede (escravos) apenas respondem quando solicitados.

Rede de comunicação multimestre Rede de comunicação onde as transferências de informações são iniciadas por qualquer nó ligado ao barramento de dados.

Run Um dos estados de uma tarefa no Processador Multitarefa AL-2005/RTMP. Indica que uma tarefa está executando.

RX Sigla usada para indicar recepção serial.

Scheduler É o gerenciador de um sistema operacional multitarefa responsável pelo escalonamento (troca de estado) das tarefas.

Semáforo Estrutura de dados da aplicação que pode ser usada pelo gerenciador de semáforos para fornecer acesso mutuamente exclusivo a recursos específicos do usuário.

Slot Uma das *n* posições usadas para armazenar elementos em uma lista circular.

Software Programas de computador, procedimentos e regras relacionadas à operação de um sistema de processamento de dados.

Sub-rede Segmento de uma rede de comunicação que interliga um grupo de equipamentos (nós) com o objetivo de isolar o tráfego local ou utilizar diferentes protocolos ou meio físicos.

Tag Nome associado a um operando ou a uma lógica que permite uma identificação resumida de seu conteúdo.

Tarefa Módulo de programa executado pelo sistema operacional de tal maneira que parece que todas as tarefas são executadas simultaneamente.

tarefa, Prioridade de uma Prioridade na qual uma tarefa é executada. Tarefas que possuem a mesma prioridade são ordenadas pela ordem na qual foram criadas.

Taskid Inteiro sem sinal atribuído a uma tarefa quando a mesma é criada e que serve como seu identificador único.

Temporizador Facilidade fornecida pelo sistema operacional que permite a medição de intervalos de tempo precisos.

temporizador, Identificador de um Inteiro sem sinal atribuído a um temporizador pelo sistema operacional e que serve como seu identificador único.

Tick de hardware Interrupção cíclica gerada por um relógio de hardware.

Tick do sistema Múltiplo do tick de hardware, de onde é derivada a unidade básica de tempo do sistema operacional. Todos os intervalos de tempo do sistema são medidos em múltiplos do tick do sistema.

Time-out Tempo preestabelecido máximo para que uma comunicação seja completada. Se for excedido procedimentos de retentiva ou diagnóstico serão ativados.

Time-Slicing Método de compartilhamento de UCP por intervalos de tempo.

Toggle Elemento que possui dois estados estáveis, trocados alternadamente a cada ativação.

Token É uma marca que indica quem é o mestre do barramento no momento.

TX Sigla usada para indicar transmissão serial.

UCP Sigla para unidade central de processamento. Controla o fluxo de informações, interpreta e executa as instruções do programa e monitora os dispositivos do sistema.

Upload Leitura do programa ou configuração do CP.

Wait Um dos estados de uma tarefa no Processador Multitarefa AL-2005/RTMP, indica que a tarefa esta esperando por algum evento.

WD Sigla para cão de guarda em inglês (watchdog). Veja circuito de cão de guarda.

Word Unidade de informação composta por 16 bits.